▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 30

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 30

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico en Informática. OTROS LENGUAJES (ADA): Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 255 09 99. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-175-4
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.
Diciembre, 1987
P.V.P. Canarias: 335,–.

INDICE

4 INFORMATICA BASICA

8 MAQUINA Z-80

11 PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

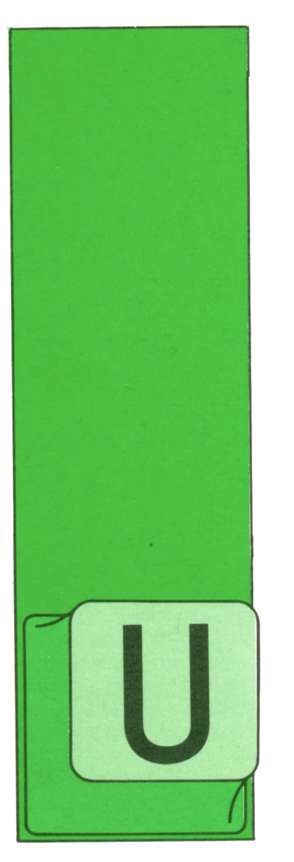
26 TECNICAS DE ANALISIS

28 TECNICAS DE PROGRAMACION

31 APLICACIONES

35 PASCAL

39 OTROS LENGUAJES



INFORMATICA BASICA

PROGRAMACION



Introducción

UNA vez que los analistas y diseñadores tienen una idea clara de cómo informatizar una empresa, si es el caso, o cómo abordar una aplicación, se empiezan a plasmar todas sus ideas en los llamados cuadernos de carga. En ellos se definen los datos que se van a usar, cuál va a ser su organización, qué tipo de ficheros se utilizarán, y se especifican las características del equipo soporte.

En la descripción de ficheros se indican los métodos de organización de cada uno (secuencial, indexado, etc.), y si forman parte de una base de datos. También se define cómo se accede a los datos que contienen de forma directa, relativa, etc.), de acuerdo a la estructura de los ficheros.

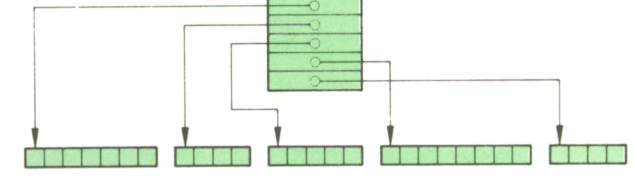




Fig. 1. Ejemplo de fichero indexado con cinco segmentos

A continuación se detallan los datos que van a constituir la entrada a los procesos, de los cuales se obtendrán los datos requeridos.

Los procesos se describirán ya en lenguaje informático, determinando cuáles son los resultados «internos», es decir, los que proporcionan datos para otros programas y no son de interés para los usuarios; y cuáles van a devolver los datos que constituyen los ficheros de salida, llamados «procesos externos». En el diseño de estos últimos es donde el programador debe poner especial atención para que los formatos sean comprensibles por el personal no informático.

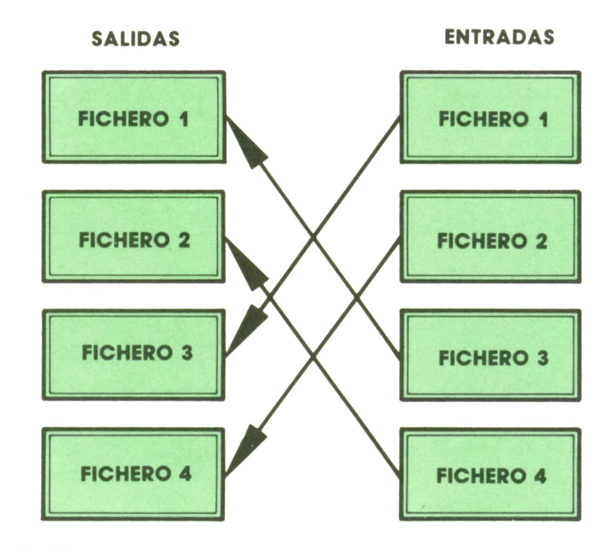


Fig. 2. La relación entre los datos de entrada y los que se esperan obtener a la salida debe quedar bien especificada en el diseño de aplicaciones.

Otro punto a desarrollar es el referente a diseño de impresos y formatos de pantalla. Los primeros están orientados al usuario final, mientras que los otros son

utilizados por los operadores de consola.

En el diseño de formatos se debe indicar claramente los campos que existen, su tipo, atributos, etc. Hay campos en los que el operador no debe introducir ningún dato (campos de salida) y otros que sí deben ser rellenados para procesarlos posteriormente (campos de entrada).

También hay que prever los posibles errores que se puedan producir a la hora de la introducción de datos, por lo que es necesario realizar validaciones a fin de evitar que se produzcan casos como el intentar rellenar campos numéricos con caracteres alfabéticos, o manipular datos o informaciones de importancia que hagan variar los datos de salida o incluso los procesos.

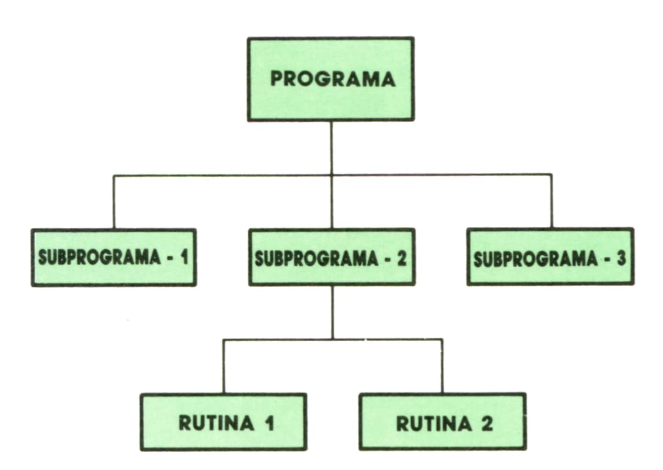


Fig. 3. Cualquier programa general puede descomponerse en un conjunto de problemas mínimo.

Programación modular

El objetivo de la programación modular es dividir un programa global en una serie de módulos más sencillos, de forma que cada uno de ellos realice una tarea única e independiente.

Esta forma de programar proporciona a los programas dos características muy importantes: flexibilidad y transportabilidad.

Flexibilidad a la hora de introducir modificaciones en una aplicación, ya que la localización de cada módulo es mucho más rápida, y la transformación en ello no tiene gran incidencia en el programa global.

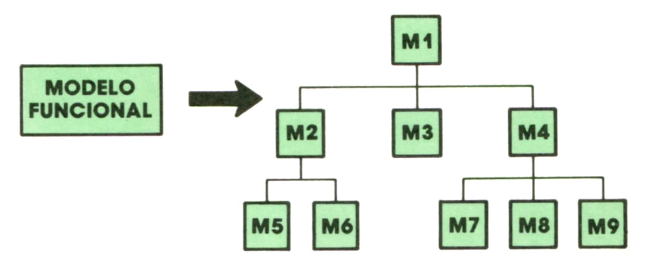




Fig. 4. Proceso de modularización.

Teniendo la gran evolución que se está produciendo actualmente en los equipos informáticos, y el constante cambio en los requerimientos por parte de los usuarios al sistema, la programación de la programación modular es importante al evitar realizar profundos cambios en la estructura de los programas, y mucho menos en los datos intermedios.

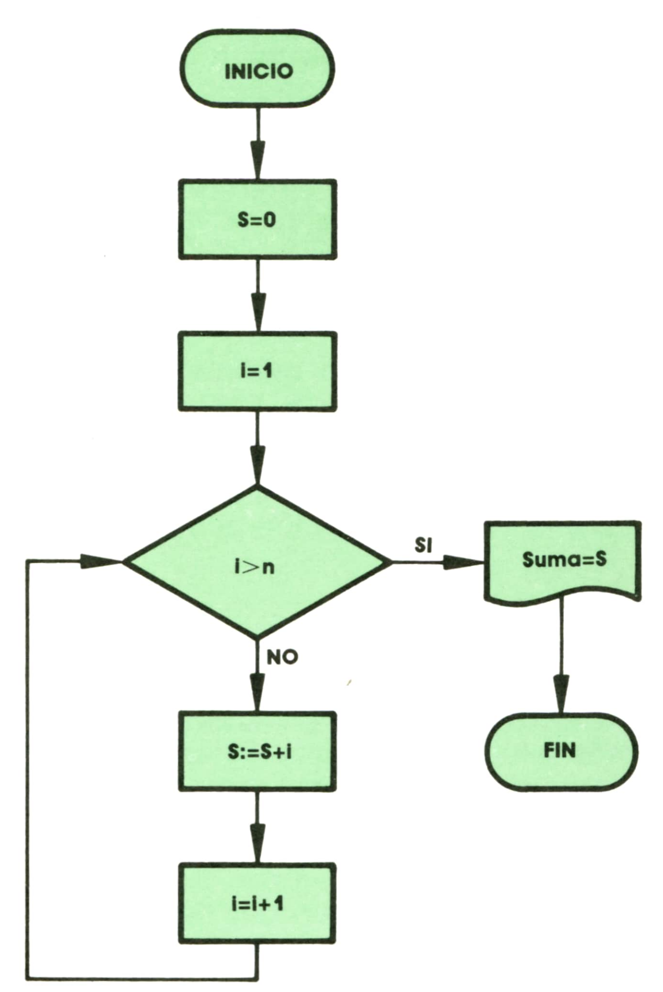




Fig. 5. Organigrama que representa el proceso a seguir para ejecutar sumatorio:

$$Y = \sum_{i=1}^{n} i$$

Este programa podría constituir un modelo de un programa mayor de cálculo, por ejemplo.

La gran ventaja de la programación modular es que al subdividir un problema en varias tareas más pequeñas y sencillas, es más fácil de escribir, leer y mantener. Cada una de estas tareas constituye un módulo fácilmente identificable.

Cada módulo tiene estructura propia, pero a su vez es perfectamente encajable en una estructura de nivel superior, de forma que no sólo dan significado a un programa, sino que también lo tienen por sí mismos.

De esta característica se deduce su **transportabilidad.** Los programas divididos en módulos son perfectamente encajables en entornos en los que no han sido diseñados, tanto físicos (ordenadores diferentes) como lógicos (programas de los cuales no formaban parte inicialmente).

El que estas características puedan darse implica la utilización de un hardware y un software adecuados.

En cuanto al hardware, se requiere la existencia de soportes de acceso directo, y controladores independientes del ordenador central.

La parte del software que más responsabilidad tiene en este caso es la relativa al sistema operativo, montadores, reubicadores, etc.

Hay que tener en cuenta que no se puede programar de esta forma con cualquier lenguaje, es necesario utilizar un lenguaje de programación estructurado que facilite la implantación de módulos y la interacción entre ellos.

Programación estructurada

Con la programación modular se hace más referencia al software que al hardware, mientras que la programación estructurada es una técnica que sirve de soporte a programas que han sido diseñados en forma modular. No hay que pensar, pues, que son dos tipos de programación diferentes, ya que no es concebible utilizar una sin la otra.

Como ya hemos visto, hay que intentar que todo programa sea flexible para que pueda adaptarse a cambios, y transportable para que cualquier nuevo proceso pueda utilizar sus rutinas sin introducir grandes cambios. Para ello se requiere que los programas sean modulares, lo que obliga a utilizar un software de base apropiado, ya que el grado de modularidad depende del intérprete empleado. Cuanto menores sean los módulos obtenidos, mayor será la transportabilidad del programa.

En un programa estructurado se utilizan tres tipos de estructuras: lineal, de bifurcación y repetitivas.

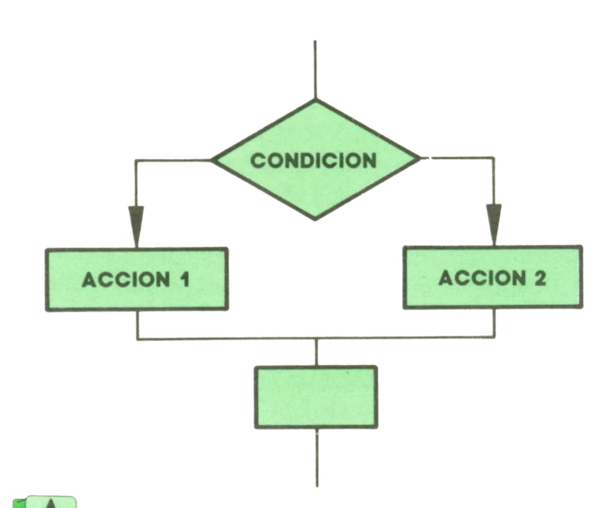


Fig. 6. Estructura alternativa, provoca la bifurcación a la realización de una de las acciones dependiendo de la condición.

La estructura lineal es la que siguen las secuencias de instrucciones dentro de un programa.

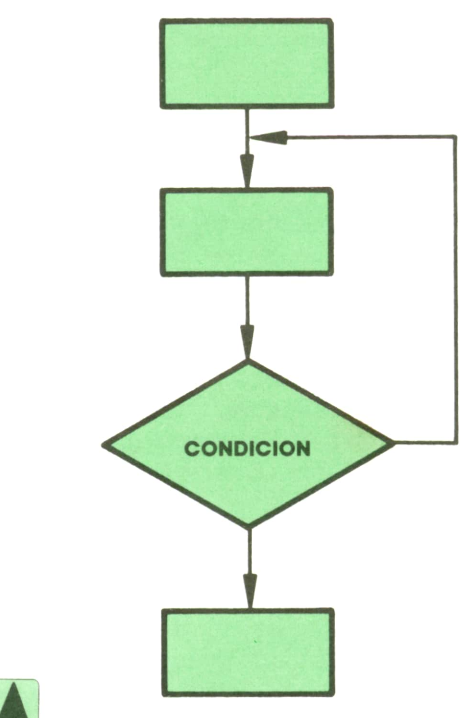


Fig. 7. Estructura repetitiva.

Una bifurcación se produce cuando existen varias acciones a realizar, dependiendo de que se cumplan o no ciertas condiciones.

Una sentencia de repetición es aquella que ejecuta un conjunto de instrucciones repetidamente hasta que se cumple cierta condición.

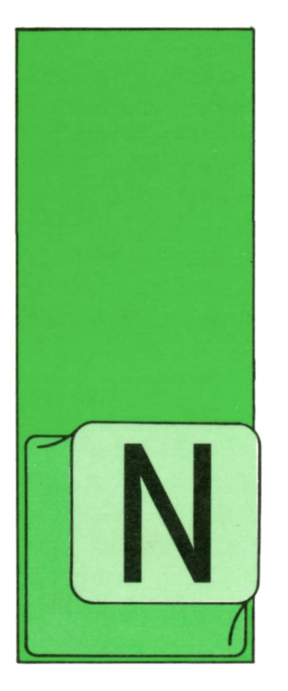
En resumen, en la programación estructurada se comienza por «modular» el programa total, y cada uno de los módulos se diseña utilizando los tres tipos de estructuras señaladas anteriormente.

Cuando un bloque de instrucciones se maneja como si fuera una única instrucción, se habla de **procedimientos.**

MAQUINA Z-80

SPECTRUM, AMSTRAD, MSX

Interrupciones



ORMALMENTE el Z-80 estará ejecutando un determinado programa; mientras tanto ignora cualquiera cosa que no sea este programa. Por ello, si queremos que deje el programa que está ejecutando en ese instante debemos darle una señal, o interrupción, que le indique que debe dejar lo que está haciendo para pasar a ocuparse de una nueva tarea.

Este proceso es similar al que ocurre cuando estamos leyendo un libro y nos llaman por teléfono. Dejamos lo que estábamos haciendo, leer un libro, marcamos con una señal el libro y procedemos a atender la interrupción, coger el teléfono. Una vez finalizada la llamada continuaremos la lectura del libro en donde lo dejamos indicado.

El proceso por el que el Z-80 atiende una interrupción es muy parecido. Primeramente guarda la dirección de la instrucción que estaba ejecutando en el momento de producirse la interrupción, de forma similar a como cuando se produce una llamada a una subrutina; ejecuta la rutina de atención a la interrupción y regresa a continuar la ejecución donde lo dejó.

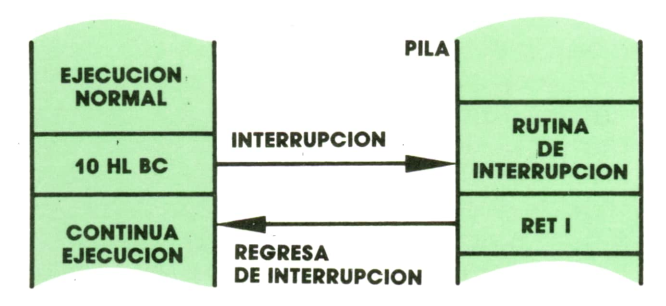




Fig. 1. Ejemplo de interrupción.

Normalmente la función de una interrupción es llevar a cabo una operación de entrada-salida. Un dispositivo como, por ejemplo, un teclado, que desconoce cuándo puede llegar un dato, interrumpe al procesador para darle el dato recién llegado. Después de esto el procesador continúa su ejecución como si esta operación no se hubiese llevado a cabo.

Interrupciones no enmascarables

Este tipo de interrupción no puede inhibirse. Por ello se reserva para periféricos, o funciones muy importantes, y que no atenderla pueda ocasionar un fallo importante del sistema.

Cuando se ejecuta dicha interrupción del sistema salta a las instrucciones situadas a partir de la posición $66 (el símbolo $ indica que el número que viene detrás está en notación hexadecimal).

Para retomar de la ejecución de una interrupción de este tipo se debe regresar con la instrucción RETN.

Existe también la posibilidad de ejecutar una interrupción no enmascarable desde programa. Esto suele tener utilidad para llamar a ciertas rutinas de la ROM, tal como se realiza el sistema operativo CP/M, o en el caso de que se hubiese producido un desastre, reinicializar el sistema. Esta es la instrucción:

RST N

donde *n* es alguna de las siguientes direcciones: $00 , $08, $10, $20, $30, $38.



Interrupciones enmascarables

Puesto que la ejecución de una interrupción puede verse interrumpida por

		Indicadores				Códigos									
Código mnemotécnico	Operación simbólica	S Z	H	P/V	N C	76	543	210	Hex	N.º de bytes	N.º de ciclos M	N.º de estados T	tarios		
DI*	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011 F3	1	1	4		
EI*	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011 FB	1	1	4		
IM 0	Selecciona modo 0 para las interrupciones	•	•	X	•	X	•	•	•	11 101 101 ED 01 000 110 46	2	2	8		
IM 1	Selecciona modo 1 para las interrupciones	•	•	X	•	X	•	•	•	11 101 101 ED 01 010 110 56	2	2	8		
IM 2	Selecciona modo 2 para las interrupciones	•	•	X	•	X	•	•	•	11 101 101 ED 01 011 110 5E	2	2	8		
RETI	Retorno de una interrupción	•	•	X	•	X	•	•	•	11 101 101 ED 01 001 101 4D	2	4	14		
RETN[1]	Retorno de una interrupción no enmascarable	•	•	X	•	X	•	•	•	11 101 101 ED 01 000 101 45	2	4	14		
RST p	$(SP-1 \leftarrow PC_H$ $(SP-2 \leftarrow PC_l$ $PC_H \leftarrow 0$ PC_l	•	•	X	•	X	•	•	•	11 t 111	1	3	11	t 000 001 010 011 100 101 110 111	p 00H 08H 10H 18H 20H 28H 30H 38H

Notas: [1] RETN carga $IFF_2 \rightarrow IFF_1$.


Fig. 2.

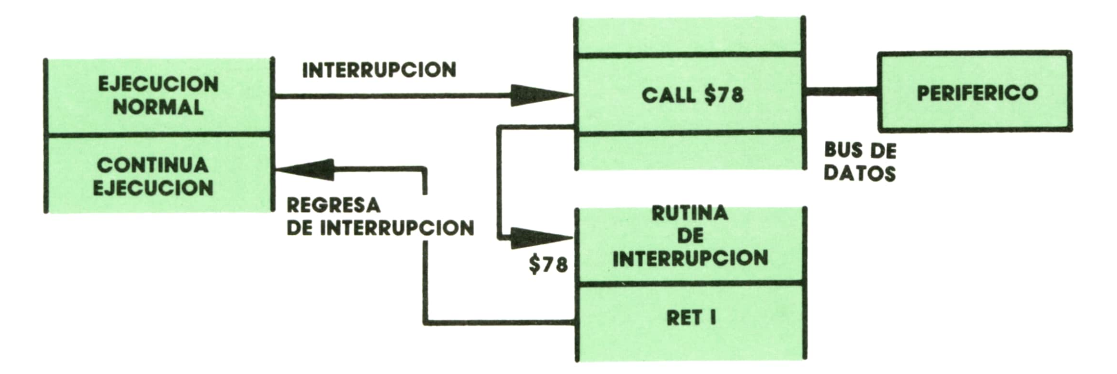



Fig. 3. Modo 0 de interrupción.

otra interrupción, es necesario prevenir algún medio para que en ciertos casos las interrupciones puedan ser inhibidas.

Esto se ve claro en el caso de estar escribiendo un dato en una unidad de disco, en el que el sincronismo es fundamental y el teclado pide una interrupción. En este caso se ejecutaría la rutina de atención al teclado, ignorando a la unidad de disco, siendo esto fatal, ya que al regresar a la rutina de atención al disco ya no escribiría sobre la pista deseada, sino sobre otra. Por esto se indicó que la interrupción no enmascarable sólo se usa para los casos muy importantes.

Para el control del permiso de inhibir las instrucciones no enmascarables existen dos instrucciones:

— EI: permite interrupciones enmascarables.

— DI: no permite interrupciones enmascarables.

Dentro de este tipo de instrucciones se pueden programar para que funcionen en 3 modos diferentes:

Modo 0:

En este caso no pasa a ejecutarse ningún programa situado en la memoria principal, sino que es el dispositivo que ha ocasionado la interrupción el que indicará el código de la siguiente instrucción a ejecutar, pudiendo ser cualquiera de las disponibles, pero suele ser una instrucción del tipo CALL.

Para activar este modo debe ejecutarse anteriormente la instrucción: IMO.

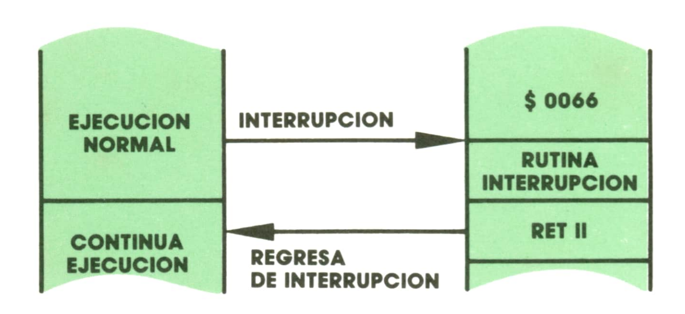




Fig. 4. Modo 1 de interrupciones

Modo 1:
Al recibirse ejecutará las instrucciones situadas a partir de la posición $38. Es análoga al funcionamiento de la interrupción no enmascarable, pero ejecutándose a partir de la posición $38 en vez de la $66.

Se activa ejecutando IM1.

Modo 2:
Este modo permite ejecutar una instrucción CALL con direccionamiento indirecto a cualquier posición de memoria con un sólo byte suministrado por el periférico que ocasiona la interrupción.

Para ello suele ser habitual mantener una tabla de direcciones de 16 bits de atención a las diferentes interrupciones.

Al ejecutarse la instrucción el argumento del salto indirecto se proporciona los 8 bits más significativos por el contenido del registro I, que deberá ser cargado anteriormente con el valor deseado. Los otros 8 bits, los menos significativos, serán proporcionados por el periférico. (En realidad, son 7 bits, ya que el bit menos significativo siempre es cero, al situarse las tablas a partir de posiciones pares de memoria.)

Se activa con IM2.

Para regresar de todas estas interrupciones se deberá ejecutar la instrucción de retorno RETI.

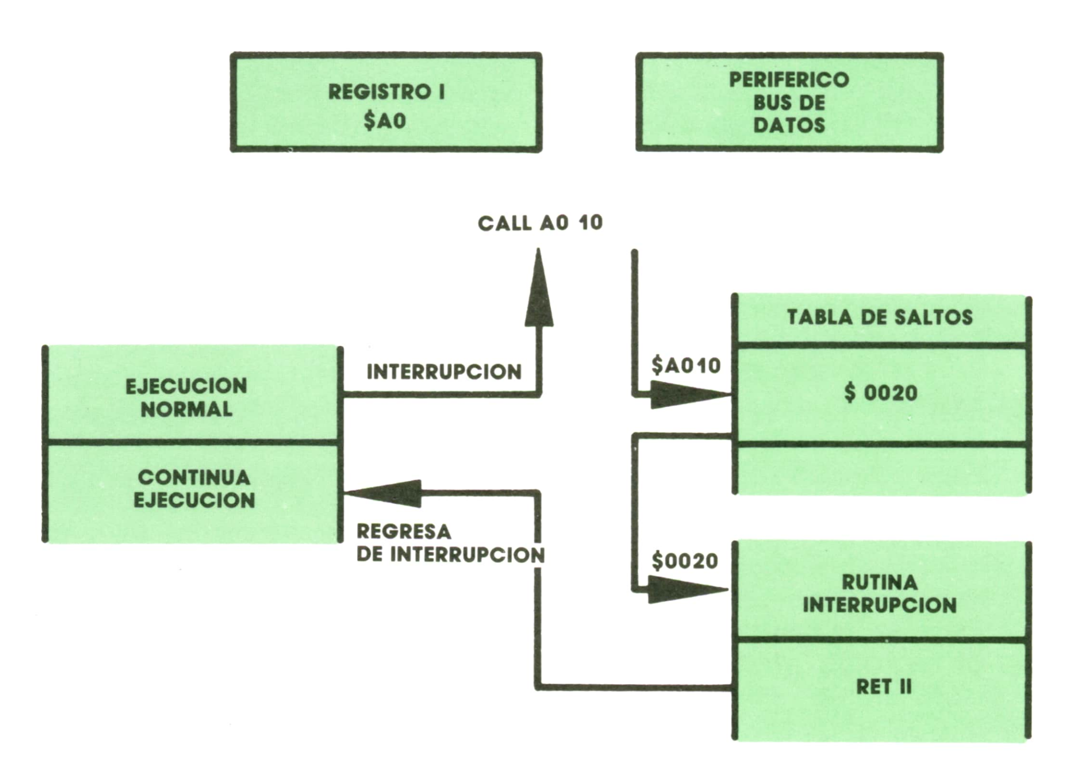
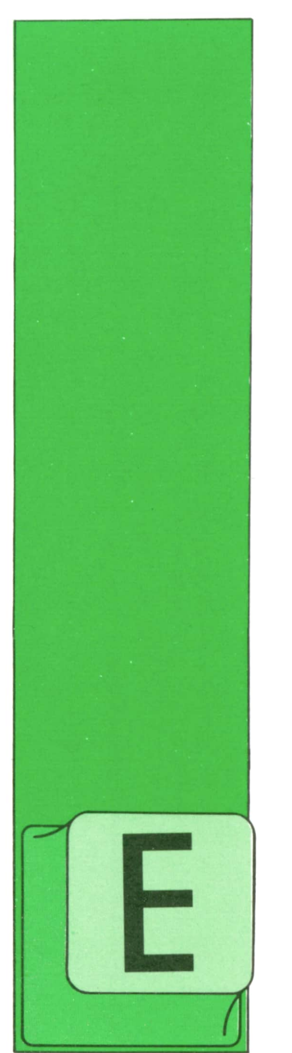




Fig. 5. Modo 2 de interrupción.

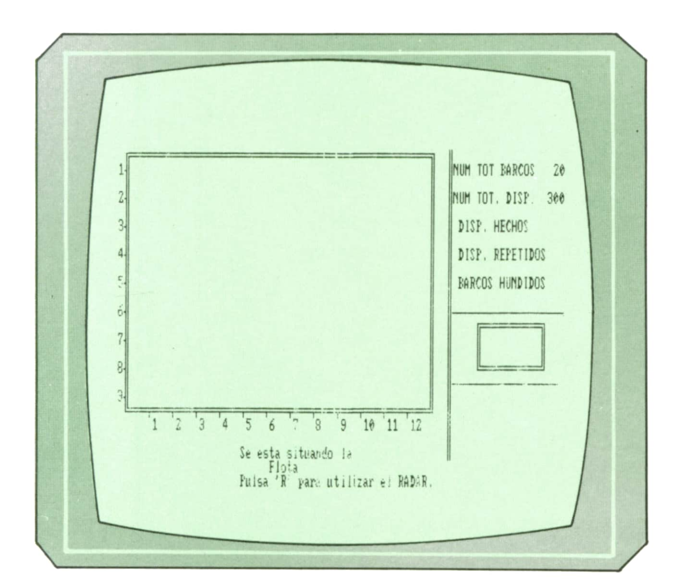
PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



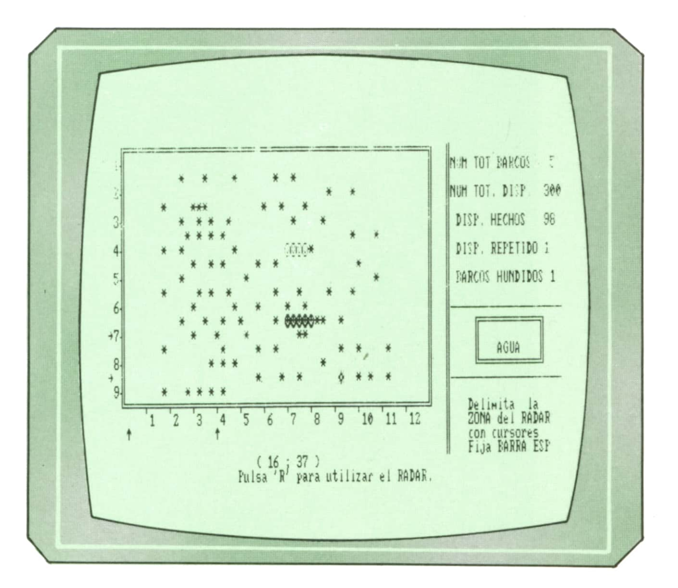
Programa: Ataque marino para IBM

L juego de los barquitos es uno de los más famosos del mundo y muchos estudiantes se entrenan con bastante frecuencia en horas de clase. El programa que vamos a ver a continuación nos permitirá jugar a los barquitos, pero de una nueva forma.




El ordenador colocando la flota.

En este juego el usuario juega contra el ordenador, pero el ordenador no juega contra el usuario. Al principio del juego se ha de decir cuántos barcos va a haber en el cuadrante y cuántos disparos podemos hacer. El juego consiste en destruir toda la flota que coloque el ordenador con el mínimo número de disparos.




El juego en plena ejecución.

Como acertar a un barco no es tarea fácil, nuestro computador dispone de RADAR que nos indicará cuántos impactos hay en una zona que nosotros mismos delimitaremos.

```
1000 REM ***********************************************
1010 REM *                                             *
1020 REM *    AAAA  TTTTT  AAAA   QQQQ  U    U EEEEEE  *
1030 REM *   A    A   T   A    A Q    Q U    U E       *
1040 REM *   A    A   T   A    A Q    Q U    U E       *
1050 REM *   A    A   T   A    A Q    Q U    U EEEE    *
1060 REM *   AAAAAA   T   AAAAAA Q  Q Q U    U E       *
1070 REM *   A    A   T   A    A Q   QQ U    U E       *
1080 REM *   A    A   T   A    A  QQQQ   UUUU  EEEEEE  *
1090 REM *                                             *
1100 REM *                                             *
1110 REM *    M    M  AAAA  RRRRR III N    N  OOOO     *
1120 REM *    MM  MM A    A R    R I  NN   N O    O    *
1130 REM *    M MM M A    A R    R I  N N  N O    O    *
1140 REM *    M    M A    A RRRRR  I  N NN N O    O    *
1150 REM *    M    M AAAAAA R R    I  N  N N O    O    *
1160 REM *    M    M A    A R  R   I  N   NN O    O    *
1170 REM *    M    M A    A R   R III N    N  OOOO     *
1180 REM *                                             *
1190 REM ***********************************************
1200 REM
1210 REM **************************
1220 REM * (c) Ed. Siglo Cultural *
1230 REM * (c) 1987.              *
1240 REM **************************
1250 REM
1260 REM ******************
1270 REM * INICIALIZACION *
1280 REM ******************
1290 REM
1300 KEY OFF
1310 CLS
1320 DIM PTO(100,100)
1330 DIM LGB(100)
1340 LOCATE 10,7
1350 PRINT"Dime la cantidad de barcos que componen la flota (5 - 20)"
1360 PRINT:PRINT "===> ";
1370 INPUT B$
1380 IF VAL(B$)>20 OR VAL(B$)<5 OR VAL(B$)<>INT(VAL(B$))THEN 1460
1390 FOR I=1 TO LEN (B$)
1400    LET C$=MID$(B$,I,1)
1410    IF C$<"0" OR C$>"9" THEN LET I=LEN(B$):GOTO 1460
1420 NEXT I
1430 LET NB=VAL(B$)
1440 CLS
1450 GOTO 1500
1460 BEEP
1470 LOCATE 12,41
1480 PRINT SPACE$(LEN(B$))
1490 GOTO 1340
1500 LOCATE 10,7
1510 PRINT"Dime la cantidad de disparos que tendr s (50 - 300)"
1520 PRINT:PRINT "===> ";
1530 INPUT B$
1540 IF VAL(B$)>300 OR VAL(B$)<50 OR VAL(B$)<>INT(VAL(B$)) THEN GOTO 1620
1550 FOR I=1 TO LEN (B$)
1560    LET C$=MID$(B$,I,1)
1570    IF C$>"9" OR C$<"0" THEN LET I=LEN(B$):GOTO 1620
1580 NEXT I
1590 LET NUMDISP=VAL (B$)
1600 CLS
1610 GOTO 1670
1620 BEEP
1630 LOCATE 12,41
1640 PRINT SPACE$(LEN(B$))
1650 GOTO 1500
1660 REM
```

```
1670 REM ******************************************
1680 REM *****   CREACION PANTALLA DE JUEGO   *****
1690 REM ******************************************
1700 REM
1710 LOCATE 24,25
1720 PRINT "Pulsa 'R' para utilizar el RADAR.";
1730 GOSUB 3170
1740 REM
1750 REM *************************************
1760 REM *****   UBICACION DE LA FLOTA   *****
1770 REM *************************************
1780 REM
1790 LOCATE 22,25
1800 PRINT"Se esta situando la"
1810 LOCATE 23,30
1820 PRINT"Flota"
1830 FOR CB=5 TO NB+4
1840    GOSUB 2330
1850 NEXT
1860 LOCATE 22,25
1870 PRINT SPACE$ (19)
1880 LOCATE 23,30
1890 PRINT SPACE$ (5)
1900 LOCATE 22,30
1910 PRINT"(Listo?"
1920 LOCATE 23,30
1930 PRINT"Pulsa una tecla"
1940 LET A$=INKEY$
1950 IF A$="" THEN GOTO 1940
1960 LOCATE 22,30
1970 PRINT SPACE$ (10)
1980 LOCATE 23,23
1990 PRINT SPACE$ (30)
2000 LOCATE 23,28
2010 PRINT"("
2020 LOCATE 23,33
2030 PRINT";"
2040 LOCATE 23,38
2050 PRINT")"
2060 REM
2070 REM *******************************
2080 REM *****   FASE DE DISPARO   *****
2090 REM *******************************
2100 REM
2110 GOSUB 4340
2120 REM
2130 REM ************************************************
2140 REM *****   MENSAJE FINAL. PERDIO EL JUGADOR   *****
2150 REM ************************************************
2160 REM
2170 LOCATE 22,1
2180 PRINT SPACE$(80)
2190 LOCATE 23,1
2200 PRINT SPACE$(80)
2210 LOCATE 22,10
2220 PRINT"Perdiste. Te quedaron  ";NB-BUN;" barcos sin hundir"
2230 GOTO 2320
2240 LOCATE 22,1
2250 PRINT SPACE$(80)
2260 LOCATE 23,1
2270 PRINT SPACE$(80)
2280 LOCATE 22,25
2290 PRINT"FELICIDADES"
2300 LOCATE 23,22
2310 PRINT"Hundiste la Flota"
2320 END
2330 LET ERD1=0
```

```
2340 LET ERD2=0
2350 LET ERD3=0
2360 LET ERD4=0
2370 LET CLB=0
2380 RANDOMIZE TIMER
2390 LET X=INT(RND*12)+4
2400 LET Y=INT(RND*46)+8
2410 LET LGB=INT(RND*4)+3
2420 LET DC=INT (RND*3)+1
2430 IF (ERD1=1 AND ERD2=1 AND ERD3=1) OR (ERD4=1 AND ERD3=1 AND ERD2=1)THEN GOT
O 2330
2440 ON DC GOTO 2450,2630,2810,2990
2450 IF ERD1=1 THEN GOTO 2420
2460 FOR N=X TO X-2 STEP -1
2470    IF PTO (N,Y)>0 THEN LET ERD1=1:GOTO 2420
2480 NEXT N
2490 FOR N=X TO X-LGB+1 STEP -1
2500    IF PTO(N,Y)>0 THEN LET LGB=CLB:GOTO 2580
2510    LET CLB=CLB+1
2520    LET PTO(N,Y)=CB
2530    LET PTO(N,Y-1)=4
2540    LET PTO(N,Y+1)=4
2550    LET PTO(N-1,Y-1)=4
2560    LET PTO(N-1,Y+1)=4
2570 NEXT N
2580 LET PTO(X+1,Y-1)=4
2590 LET PTO(X+1,Y)=4
2600 LET PTO(X+1,Y+1)=4
2610 LET PTO(X-LGB,Y)=4
2620 GOTO 3160
2630 IF ERD2=1 THEN GOTO 2420
2640 FOR N=Y TO Y-2 STEP -1
2650    IF PTO(X,N)>0 THEN LET ERD2=1:GOTO 2420
2660 NEXT N
2670 FOR N=Y TO Y-LGB+1 STEP -1
2680    IF PTO (X,N)>0 THEN LET LGB=CLB:GOTO 2760
2690    LET CLB=CLB+1
2700    LET PTO(X,N)=CB
2710    LET PTO(X-1,N)=4
2720    LET PTO(X+1,N)=4
2730    LET PTO(X-1,N-1)=4
2740    LET PTO(X+1,N-1)=4
2750 NEXT N
2760 LET PTO(X-1,Y+1)=4
2770 LET PTO(X,Y+1)=4
2780 LET PTO(X+1,Y+1)=4
2790 LET PTO(X,Y-LGB)=4
2800 GOTO 3160
2810 IF ERD3=1 THEN GOTO 2420
2820 FOR N=X TO X+2
2830    IF PTO(N,Y)>0 THEN LET ERD3=1:GOTO 2420
2840 NEXT N
2850 FOR N=X TO X+LGB-1
2860    IF PTO(N,Y)>0 THEN LET LGB=CLB:GOTO 2940
2870    LET CLB=CLB+1
2880    LET PTO(N,Y)=CB
2890    LET PTO(N,Y-1)=4
2900    LET PTO(N,Y+1)=4
2910    LET PTO(N+1,Y-1)=4
2920    LET PTO(N+1,Y+1)=4
2930 NEXT N
2940 LET PTO(X-1,Y-1)=4
2950 LET PTO(X-1,Y)=4
2960 LET PTO(X-1,Y+1)=4
2970 LET PTO(X+LGB,Y)=4
2980 GOTO 3160
2990 IF ERD4=1 THEN GOTO 2420
```

```
3000 FOR N=Y TO Y+2
3010    IF PTO(X,N)>0 THEN LET ERD4=1:GOTO 2420
3020 NEXT N
3030 FOR N=Y TO Y+LGB-1
3040    IF PTO(X,N)>0 THEN LET LGB=CLB:GOTO 3120
3050    LET CLB=CLB+1
3060    LET PTO(X,N)=CB
3070    LET PTO(X-1,N)=4
3080    LET PTO(X+1,N)=4
3090    LET PTO(X-1,N+1)=4
3100    LET PTO(X+1,N+1)=4
3110 NEXT N
3120 LET PTO(X-1,Y-1)=4
3130 LET PTO(X,Y-1)=4
3140 LET PTO(X+1,Y-1)=4
3150 LET PTO(X,Y+LGB)=4
3160 RETURN
3170 LET YD1=6
3180 LET YD2=56
3190 LET Z=205
3200 LET XD1=1
3210 GOSUB 4130
3220 LET XD1=19
3230 GOSUB 4130
3240 LET YD1=66
3250 LET YD2=75
3260 LET XD1=13
3270 GOSUB 4130
3280 LET XD1=16
3290 GOSUB 4130
3300 LET YD1=61
3310 LET YD2=79
3320 LET XD1=12
3330 LET Z=196
3340 GOSUB 4130
3350 LET XD1=17
3360 GOSUB 4130
3370 LET XD1=2
3380 LET XD2=18
3390 LET Z=186
3400 LET YD1=5
3410 GOSUB 4180
3420 LET YD1=57
3430 GOSUB 4180
3440 LET YD1=60
3450 LET XD2=22
3460 LET XD1=1
3470 GOSUB 4180
3480 LET XD1=14
3490 LET XD2=15
3500 LET YD1=65
3510 GOSUB 4180
3520 LET YD1=76
3530 GOSUB 4180
3540 LET XD1=1
3550 LET YD1=5
3560 LET YD2=57
3570 LET XD2=19
3580 GOSUB 4230
3590 LET XD1=13
3600 LET YD1=65
3610 LET YD2=76
3620 LET XD2=16
3630 GOSUB 4230
3640 LOCATE 2,61
3650 PRINT "NUM TOT BARCOS"
3660 LOCATE 2,77
```

```
3670 PRINT NB
3680 LOCATE 4,61
3690 PRINT "NUM TOT. DISP."
3700 LOCATE 4,76
3710 PRINT NUMDISP
3720 LOCATE 6,62
3730 PRINT"DISP. HECHOS"
3740 LOCATE 8,62
3750 PRINT"DISP. REPETIDOS"
3760 LOCATE 10,62
3770 PRINT"BARCOS HUNDIDOS"
3780 LET H=6
3790 LET J=56
3800 LET K=1
3810 GOSUB 4050
3820 LET K=19
3830 GOSUB 4050
3840 LET H=2
3850 LET J=18
3860 LET K=5
3870 GOSUB 4090
3880 LET K=57
3890 GOSUB 4090
3900 FOR C=2 TO 18 STEP 2
3910    LET CNF=CNF+1
3920    LOCATE C,3
3930    PRINT CNF
3940    LOCATE C,5
3950    PRINT CHR$(182)
3960 NEXT C
3970 FOR C=9 TO 53 STEP 4
3980    LET CNC=CNC+1
3990    LOCATE 20,C
4000    PRINT CNC
4010    LOCATE 19,C
4020    PRINT CHR$(209)
4030 NEXT C
4040 RETURN
4050 FOR F=H TO J
4060    LET PTO(K,F)=4
4070 NEXT F
4080 RETURN
4090 FOR F=H TO J
4100    LET PTO(F,K)=4
4110 NEXT F
4120 RETURN
4130 FOR C=YD1 TO YD2
4140    LOCATE XD1,C
4150    PRINT CHR$(Z)
4160 NEXT C
4170 RETURN
4180 FOR C=XD1 TO XD2
4190    LOCATE C,YD1
4200    PRINT CHR$(Z)
4210 NEXT C
4220 RETURN
4230 LOCATE XD1,YD1
4240 PRINT CHR$(201)
4250 LOCATE XD1,YD2
4260 PRINT CHR$(187)
4270 LOCATE XD2,YD1
4280 PRINT CHR$(200)
4290 LOCATE XD2,YD2
4300 PRINT CHR$(188)
4310 RETURN
4320 REM
```

```
4330 REM ***************************************
4340 REM *******   FASE DE DISPARO    ***********
4350 REM ***************************************
4360 REM
4370 LET CX=10
4380 LET CY=31
4390 FOR DISP=1 TO NUMDISP
4400    LOCATE CX,CY
4410    COLOR 18
4420    PRINT CHR$(15)
4430    COLOR 2
4440    A$=INKEY$:IF A$="" THEN 4440
4450    IF A$="8" THEN GOTO 4530
4460    IF A$="4" THEN GOTO 4650
4470    IF A$="2" THEN GOTO 4770
4480    IF A$="6" THEN GOTO 4890
4490    IF ASC(A$)=32 THEN GOTO 5000
4500    IF A$="R" OR A$="r" THEN GOTO 5620
4510    BEEP
4520    GOTO 4440
4530    LET CX=CX-1
4540    IF CX<2 THEN LET CX=CX+1:BEEP:GOTO 4610
4550    IF PTO(CX+1,CY)=1 THEN LET Z=42:GOTO 4590
4560    IF PTO(CX+1,CY)=2 THEN LET Z=1:GOTO 4590
4570    IF PTO(CX+1,CY)=3 THEN LET Z=2:GOTO 4590
4580    LET Z=32
4590    LOCATE CX+1,CY
4600    PRINT CHR$(Z)
4610    LOCATE CX,CY
4620    COLOR 18
4630    PRINT CHR$(15)
4640    GOTO 4980
4650    LET CY=CY-1
4660    IF CY<6 THEN LET CY=CY+1:BEEP:GOTO 4730
4670    IF PTO(CX,CY+1)=1 THEN LET Z=42:GOTO 4710
4680    IF PTO(CX,CY+1)=2 THEN LET Z=1:GOTO 4710
4690    IF PTO(CX,CY+1)=3 THEN LET Z=2:GOTO 4710
4700    LET Z=32
4710    LOCATE CX,CY+1
4720    PRINT CHR$ (Z)
4730    LOCATE CX,CY
4740    COLOR 18
4750    PRINT CHR$ (15)
4760    GOTO 4980
4770    LET CX=CX+1:IF CX>18 THEN CX=CX-1:BEEP:GOTO 4850
4780    IF CX>18 THEN CX=CX-1:BEEP:GOTO 4850
4790    IF PTO(CX-1,CY)=1 THEN LET Z=42:GOTO 4830
4800    IF PTO(CX-1,CY)=2 THEN LET Z=1:GOTO 4830
4810    IF PTO(CX-1,CY)=3 THEN LET Z=2:GOTO 4830
4820    LET Z=32
4830    LOCATE CX-1,CY
4840    PRINT CHR$ (Z)
4850    LOCATE CX,CY
4860    COLOR 18
4870    PRINT CHR$(15)
4880    GOTO 4980
4890    LET CY=CY+1:IF CY>56 THEN CY=CY-1:BEEP:GOTO 4970
4900    IF CY>56 THEN LET CY=CY-1:BEEP:GOTO 4970
4910    IF PTO(CX,CY-1)=1 THEN LET Z=42:GOTO 4960
4920    IF PTO(CX,CY-1)=2 THEN LET Z=1:GOTO 4960
4930    IF PTO(CX,CY-1)=3 THEN LET Z=2:GOTO 4960
4940    IF PTO (CX,CY-1)>24 THEN LET Z=186:GOTO 4960
4950    LET Z=32
4960    LOCATE CX,CY-1:PRINT CHR$ (Z)
4970    LOCATE CX,CY:COLOR 18:PRINT CHR$(15)
4980    LOCATE 23,29:COLOR 2:PRINT CX-1:LOCATE 23,34:PRINT CY-5:GOTO 4440
4990 REM
5000 REM *****   APRIETA GATILLO    *****
```

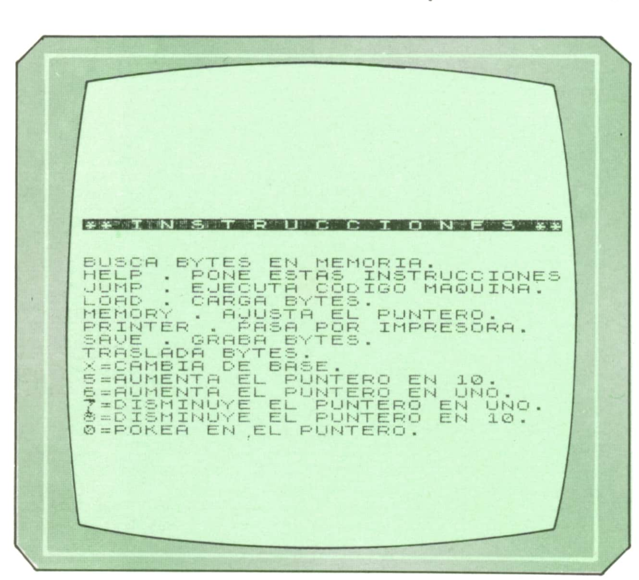
```
5010 REM
5020    LOCATE 14,66
5030    PRINT SPACE$ (10)
5040    LOCATE 15,66
5050    PRINT SPACE$ (10)
5060    IF PTO(CX,CY)=0 THEN LET PTO(CX,CY)=1:LOCATE CX,CY:PRINT CHR$(42):LOCATE
 15,69:PRINT"AGUA":GOTO 5400
5070    IF PTO(CX,CY)=1 THEN LOCATE 14,67:PRINT"REPETIDO":LOCATE 15,69:PRINT"AGU
A":LOCATE CX,CY:PRINT CHR$(42):LET CTR=CTR+1:GOTO 5400
5080    IF PTO(CX,CY)=2 THEN LOCATE 14,67:PRINT"REPETIDO":LOCATE 15,68:PRINT"TOC
ADO":LOCATE CX,CY:PRINT CHR$(1):LET CTR=CTR+1:GOTO 5400
5090    IF PTO (CX,CY)=3 THEN LOCATE 14,67:PRINT"REPETIDO":LOCATE 15,67:PRINT"HU
NDIDO":LOCATE CX,CY:PRINT CHR$ (2):LET CTR=CTR+1:GOTO 5400
5100    IF PTO(CX,CY)=4 THEN LET PTO(CX,CY)=1:LOCATE CX,CY:PRINT CHR$(42):LOCATE
 15,69:PRINT"AGUA":GOTO 5400
5110    LET PTO(CX,CY)=2
5120    LOCATE CX,CY
5130    PRINT CHR$(1)
5140    FOR I=1 TO 7
5150       IF PTO(CX-I,CY)=4 OR PTO(CX-I,CY)=1 THEN GOTO 5190
5160       IF PTO(CX-I,CY)=2 THEN GOTO 5180
5170       GOTO 5380
5180    NEXT I
5190    FOR I=1 TO 7
5200       IF PTO (CX,CY-I)=4 OR PTO(CX,CY-I)=1 THEN GOTO 5240
5210       IF PTO(CX,CY-I)=2 THEN GOTO 5230
5220       GOTO 5380
5230    NEXT I
5240    FOR I=1 TO 7
5250       IF PTO(CX+I,CY)=4 OR PTO(CX+I,CY)=1 THEN GOTO 5290
5260       IF PTO(CX+I,CY)=2 THEN GOTO 5280
5270       GOTO 5380
5280    NEXT I
5290    FOR I=1 TO 7
5300       IF PTO (CX,CY+I)=4 OR PTO (CX,CY+I)=1 THEN GOTO 5340
5310       IF PTO (CX,CY+I)=2 THEN GOTO 5330
5320       GOTO 5380
5330    NEXT I
5340    LOCATE 14,67
5350    PRINT"HUNDIDO"
5360    LET BUN=BUN+1
5370    GOTO 5460
5380    LOCATE 14,68
5390    PRINT"TOCADO"
5400    LOCATE 6,76:PRINT DISP
5410    LOCATE 8,76:PRINT CTR
5420    LOCATE 10,77:PRINT BUN
5430    IF BUN=NB THEN GOTO 2240
5440 NEXT DISP
5450 RETURN
5460 LOCATE CX,CY
5470 PRINT CHR$(2)
5480 LET PTO (CX,CY)=3
5490 FOR I=1 TO 6
5500    IF PTO(CX-I,CY)=2 THEN LET PTO(CX-I,CY)=3:LOCATE CX-I,CY:PRINT CHR$(2) E
LSE GOTO 5520
5510 NEXT I
5520 FOR I=1 TO 6
5530    IF PTO(CX,CY-I)=2 THEN LET PTO(CX,CY-I)=3:LOCATE CX,CY-I:PRINT CHR$(2) E
LSE GOTO 5550
5540 NEXT I
5550 FOR I=1 TO 6
5560    IF PTO(CX+I,CY)=2 THEN LET PTO(CX+I,CY)=3:LOCATE CX+I,CY:PRINT CHR$(2) E
LSE GOTO 5580
5570 NEXT I
5580 FOR I=1 TO 6
5590    IF PTO(CX,CY+I)=2 THEN LET PTO(CX,CY+I)=3:LOCATE CX,CY+I:PRINT CHR$(2) E
LSE GOTO 5610
```

```
5600 NEXT I
5610 GOTO 5400
5620 LOCATE 19,64:PRINT SPACE$(14)
5630 LOCATE 20,64:PRINT SPACE$(14)
5640 LOCATE 21,64:PRINT SPACE$(14)
5650 LOCATE 22,64:PRINT SPACE$(14)
5660 LOCATE 19,64:PRINT"Delimita  la"
5670 LOCATE 20,64:PRINT"ZONA del RADAR"
5680 LOCATE 21,64:PRINT"con cursores"
5690 LOCATE 22,64:PRINT"Fija BARRA ESP"
5700 FOR L=1 TO 19
5710    LOCATE L,3:PRINT" "
5720 NEXT L
5730 FOR L=5 TO 57
5740    LOCATE 21,L:PRINT" "
5750 NEXT L
5760 LET RX1=2
5770 LET RX2=18
5780 LET RY1=6
5790 LET RY2=56
5800 LET DPB=0
5810 LOCATE RX1,3:COLOR 18:PRINT CHR$(26):COLOR 2
5820 A$=INKEY$:IF A$="" THEN GOTO 5820
5830 IF A$="2" THEN IF RX1<17 THEN GOTO 5870 ELSE GOTO 5860
5840 IF A$="8" THEN IF RX1>2 THEN GOTO 5890 ELSE GOTO 5860
5850 IF ASC(A$)=32 THEN GOTO 5910
5860 BEEP:GOTO 5820
5870 LOCATE RX1,3:PRINT " "
5880 LET RX1=RX1+1:GOTO 5810
5890 LOCATE RX1,3:PRINT " "
5900 LET RX1=RX1-1:GOTO 5810
5910 LOCATE RX1,3:PRINT CHR$(26)
5920 LOCATE RX2,3:COLOR 18:PRINT CHR$(26):COLOR 2
5930 A$=INKEY$:IF A$="" THEN GOTO 5930
5940 IF A$="2" THEN IF RX2<17 THEN GOTO 5980 ELSE GOTO 5970
5950 IF A$="8" THEN IF RX2>RX1+1 THEN GOTO 6000 ELSE GOTO 5970
5960 IF ASC(A$)=32 THEN GOTO 6020
5970 BEEP:GOTO 5930
5980 LOCATE RX2,3:PRINT " "
5990 LET RX2=RX2+1:GOTO 5920
6000 LOCATE RX2,3:PRINT " "
6010 LET RX2=RX2-1:GOTO 5920
6020 LOCATE RX2,3:PRINT CHR$(26)
6030 LOCATE 21,RY1:COLOR 18:PRINT CHR$(24):COLOR 2
6040 A$=INKEY$:IF A$="" THEN GOTO 6040
6050 IF A$="6" THEN IF RY1<55 THEN GOTO 6090 ELSE GOTO 6080
6060 IF A$="4" THEN IF RY1>6 THEN GOTO 6110 ELSE GOTO 6080
6070 IF ASC(A$)= 32 THEN GOTO 6130
6080 BEEP:GOTO 6040
6090 LOCATE 21,RY1:PRINT " "
6100 LET RY1=RY1+1:GOTO 6030
6110 LOCATE 21,RY1:PRINT " "
6120 LET RY1=RY1-1:GOTO 6030
6130 LOCATE 21,RY1:PRINT CHR$(24)
6140 LOCATE 21,RY2:COLOR 18:PRINT CHR$(24):COLOR 2
6150 A$=INKEY$:IF A$="" THEN GOTO 6150
6160 IF A$="6" THEN IF RY2<56 THEN GOTO 6200 ELSE GOTO 6190
6170 IF A$="4" THEN IF RY2>RY1+1 THEN GOTO 6220 ELSE GOTO 6190
6180 IF ASC(A$)=32 THEN GOTO 6240
6190 BEEP:GOTO 6150
6200 LOCATE 21,RY2:PRINT " "
6210 LET RY2=RY2+1:GOTO 6140
6220 LOCATE 21,RY2:PRINT" "
6230 LET RY2=RY2-1:GOTO 6140
6240 LOCATE 21,RY2:PRINT CHR$(24)
6250 LOCATE 23,63:PRINT"CONFORME S/N":COLOR 18:LOCATE 23,76:PRINT"?":COLOR 2
6260 A$=INKEY$:IF A$="" THEN GOTO 6260
```

```
6270 IF A$="S" OR A$="s" THEN LOCATE 23,72:PRINT SPACE$(7):GOTO 6300
6280 IF A$="N" OR A$="n" THEN LOCATE 23,61:PRINT SPACE$(19):GOTO 5700
6290 BEEP:GOTO 6250
6300 FOR Y=RY1 TO RY2
6310    FOR X=RX1 TO RX2
6320       IF PTO(X,Y)>4 THEN LET DPB=DPB+1
6330    NEXT X
6340 NEXT Y
6350 LOCATE 18,61:PRINT SPACE$(19)
6360 LOCATE 19,61:PRINT SPACE$(19)
6370 LOCATE 20,61:PRINT SPACE$(19)
6380 LOCATE 21,61:PRINT SPACE$(19)
6390 LOCATE 22,61:PRINT SPACE$(19)
6400 LOCATE 23,61:PRINT SPACE$(19)
6410 LOCATE 19,66:PRINT"En esta ZONA"
6420 LOCATE 20,66:PRINT"Hay"
6430 LOCATE 21,66:PRINT"Posible IMPC"
6440 LOCATE 20,71:PRINT DPB
6450 GOTO 4440
6460 LOCATE 7,21:PRINT"Se est  situando la"
6470 LOCATE 9,28:PRINT"Flota"
6480 LOCATE 12,29:PRINT"Listo"
6490 LOCATE 14,22:PRINT"Para el Combate  S/N"
6500 LET DC=INT(RND*3)+1
6510 PRINT DC,
6520 GOTO 6500
```

Programa: Monitor de código máquina para Spectrum

Muchas veces es necesario trabajar con los Bytes de la memoria sin meternos en un desensamblador complicado. A veces sólo necesitamos ver el contenido de una zona de memoria, cambiar algunos Bytes, mover un bloque de una parte de la memoria a otro, etc.





Comandos de programa.

Con el programa que vamos a ver a continuación podremos hacer esto y mucho más. Las funciones que podemos realizar son las siguientes:

— Buscar Bytes en memoria.

— Ver las instrucciones (pulsando la H).

— Ejecutar un programa en código máquina.

— Cargar Bytes desde el casete.

— Ajustar el puntero de visualización.

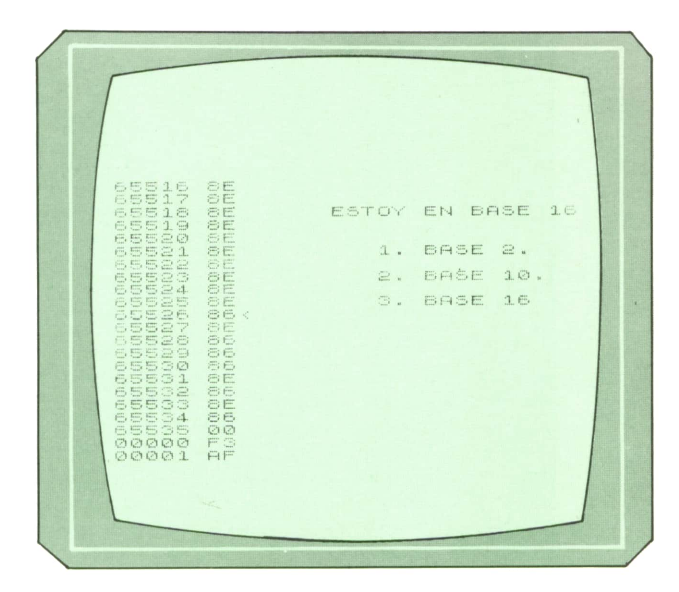
— Imprimir una serie de Bytes por impresora.

— Trasladar Bytes de una zona a otra.

— Cambiar el contenido de una posición de memoria.

— Cambiar de base de numeración.

```
00070  00110100
00071  01000000
00072  11000101ESTOY EN BASE 2
00073  11010101
00074  11001101
00075  10111111
00076  00000010
00077  11010001
00078  11000001
00079  11100001
00080  11110001 <
00081  11111011
00082  11001001
00083  11100001
00084  01101110
00085  11111101
00086  01110101
00087  00000000
00088  11101101
00089  01111011
00090  00111101
00091  01011100
```





Cambiado de base.

```
1000 REM **********************
1010 REM * MONITOR DE MEMORIA *
1020 REM **********************
1030 REM
1040 REM *******************************
1050 REM *                             *
1060 REM * (c) Ediciones Siglo Cultural *
1070 REM *                             *
1080 REM * (c) 1987                    *
1090 REM *                             *
1100 REM *******************************
1110 REM
1120 REM ******************
1130 REM * INICIALIZACION *
1140 REM ******************
1150 REM
1160 GOSUB 3210
1170 LET BAS=10
1180 POKE 23296,0
1190 POKE 23658,8
1200 LET MP=0
1210 LET MOD=2
1220 REM
1230 REM **********************
1240 REM * PROGRAMA PRINCIPAL *
1250 REM **********************
1260 REM
1270 CLS
1280 PRINT AT 2,15;"ESTOY EN BASE ";BAS
1290 GOSUB 1480
1300 LET K$=INKEY$
1310 IF K$="" THEN GOTO 1300
1320 IF K$="6" THEN LET MP=MP+(1 AND MP<65535 )-(MP AND MP=65535 ):GOSUB 1480:GO
TO 1300
1330 IF K$="7" THEN LET MP=MP-(1 AND MP>0)+(65535 -MP AND MP=0):GOSUB 1480:GOTO
1300
1340 IF K$="5" THEN LET MP=MP+(10 AND MP+10<65535 )-(10-65536! AND MP+10>65535 )
:GOSUB 1480:GOTO 1300
1350 IF K$="8" THEN LET MP=MP-(10 AND MP-10>0)+(-10+65526! AND MP-10<0):GOSUB 14
80:GOTO 1300
1360 IF K$="0" THEN INPUT "VALOR A POKEAR = ";P:POKE MP,P:GOTO 1270
1370 IF K$="H" THEN GOSUB 3520:GOTO 1270
```

```
1380 IF K$="B" THEN GOSUB 2010:GOTO 1300
1390 IF K$="N" AND PEEK(29900)<>0 THEN GOSUB 2180:GOTO 1300
1400 IF K$="J" THEN GOSUB 2250:GOTO 1270
1410 IF K$="L" THEN GOSUB 2350:GOTO 1270
1420 IF K$="S" THEN GOSUB 2560:GOTO 1270
1430 IF K$="M" THEN GOSUB 2680:GOTO 1270
1440 IF K$="X" THEN GOSUB 2770:GOTO 1270
1450 IF K$="P" THEN GOSUB 1720:GOTO 1270
1460 IF K$="T" THEN GOSUB 2920:GOTO 1270
1470 GOTO 1300
1480 REM
1490 REM ************************
1500 REM * DISPLAY DE LA MEMORIA *
1510 REM ************************
1520 REM
1530 PRINT AT 0,0;
1540 FOR A=MP-10 TO MP+11
1550   LET B=A
1560   IF A>65535  THEN LET A=A-65536
1570   IF A<0 THEN LET A=65536!+A
1580   LET N$=STR$(A)
1590   LET P$=STR$(PEEK(A))
1600   IF MOD=1 THEN GOSUB 3100
1610   IF LEN(N$)<5 THEN LET N$="0"+N$:GOTO 1610
1620   IF MOD=2 THEN IF LEN(P$)<3 THEN LET P$="0"+P$:GOTO 1620
1630   IF MOD=3 THEN LET NUM=VAL(P$):GOSUB 1890
1640   PRINT " ";N$;"  ";P$
1650   LET A=B
1660 NEXT A
1670 PRINT AT 10,0;">"
1680 IF MOD=1 THEN PRINT AT 10,15;"<"
1690 IF MOD=2 THEN PRINT AT 10,10;"<"
1700 IF MOD=3 THEN PRINT AT 10,9;"<"
1710 RETURN
1720 REM
1730 REM ************************
1740 REM * SALIDA POR IMPRESORA *
1750 REM ************************
1760 REM
1770 INPUT "DESDE LA DIR. ";I
1780 IF I<0 THEN GOTO 1770
1790 INPUT "HASTA LA DIR. ";F
1800 IF F<I OR F>65535  THEN GOTO 1790
1810 LET A=MP
1820 FOR M=I+10 TO F STEP 22
1830    LET MP=M
1840    GOSUB 1480
1850    COPY
1860 NEXT M
1870 LET MP=A
1880 RETURN
1890 REM
1900 REM **********************************
1910 REM * PASO DE DECIMAL A HEXADECIMAL *
1920 REM **********************************
1930 REM
1940 LET P$=""
1950 FOR C=1 TO 0 STEP -1
1960    LET N=INT(NUM/16^C)
1970    LET NUM=NUM-N*16^C
1980    LET P$=P$+(STR$(N) AND N<10)+("A" AND N=10)+("B" AND N=11)+("C" AND N=12
)+("D" AND N=13)+("E" AND N=14)+("F" AND N=15)
1990 NEXT C
2000 RETURN
2010 REM
2020 REM *********************
2030 REM * BUSQUEDA DE BYTES *
2040 REM *********************
```

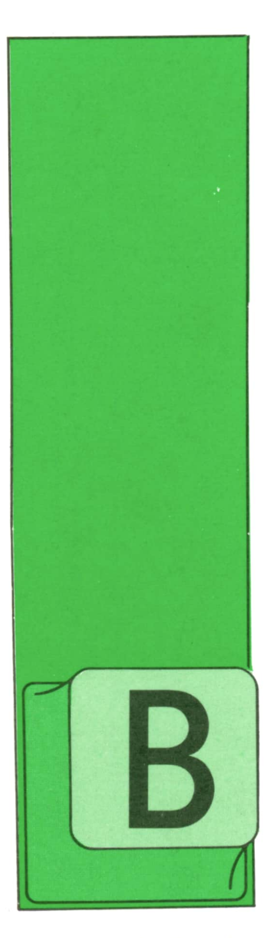
```
2050 REM
2060 POKE 23728,0
2070 POKE 23729,0
2080 INPUT "CUANTOS BUSCO? (1-5) ";B
2090 IF B>5 OR B<1 THEN GOTO 2080
2100 POKE 29900,B
2110 FOR A=1 TO B
2120    CLS
2130    PRINT "DAME EL NUMERO ";A
2140    INPUT "NUMERO > ";C
2150    IF C>255 OR C<0 THEN GOTO 2140
2160    POKE 29900+A,C
2170 NEXT A
2180 LET A=USR 29500
2190 RANDOMIZE A+1
2200 POKE 23728,PEEK(23670)
2210 POKE 23729,PEEK(23671)
2220 LET MP=A
2230 GOSUB 1480
2240 RETURN
2250 REM
2260 REM ************************************
2270 REM * SALTO A UNA DIRECCION DE MEMORIA *
2280 REM ************************************
2290 REM
2300 INPUT "A QUE DIRECCION SALTO? (0-65535) ";J
2310 IF J<0 OR J>65535  THEN GOTO 2300
2320 INPUT "ESTAS SEGURO? (S/N) ";A$
2330 IF A$(1)="S" OR A$(1)="s" THEN PRINT USR J
2340 GOTO 3750
2350 REM
2360 REM ***********************
2370 REM * CARGAR BYTES (LOAD) *
2380 REM ***********************
2390 REM
2400 GOSUB 2450
2410 IF A$(1)="M" OR A$(1)="m" THEN LOAD *"M";1;N$CODE D:RETURN
2420 IF A$(1)<>"C" AND A$(1)<>"c" THEN GOTO 2400
2430 LOAD D$CODE D
2440 RETURN
2450 REM
2460 REM ************************
2470 REM * CASETE O MICRODRIVE? *
2480 REM ************************
2490 REM
2500 INPUT FLASH 1;"C";FLASH 0;"ASETE O ";FLASH 1;"M";FLASH 0;"ICRO? ";LINE A$
2510 INPUT "NOMBRE? (0-10 CHRS.) ";N$
2520 IF LEN(N$)>10 THEN GOTO 2510
2530 INPUT "DIRECCION? ( >30000 ) ";D
2540 IF D<30000 OR D>65536  THEN GOTO 2530
2550 RETURN
2560 REM
2570 REM **********************
2580 REM * GRABAR BYTES (SAVE) *
2590 REM **********************
2600 REM
2610 GOSUB 2450
2620 INPUT "LONGITUD? ";L
2630 IF L>35535! OR L<0 THEN GOTO 2620
2640 IF A$(1)="M" OR A$(1)="m" THEN SAVE *"M";1;N$CODE D,L:RETURN
2650 IF A$(1)<>"C" AND A$(1)<>"c" THEN GOTO 2610
2660 SAVE N$CODE D,L
2670 RETURN
2680 REM
2690 REM ******************************
2700 REM * CAMBIAR PUNTERO DE MEMORIA *
2710 REM ******************************
2720 REM
```

```
2730 INPUT "MEMORY = ";M
2740 IF M>65535  OR M<0 THEN GOTO 2730
2750 LET MP=M
2760 RETURN
2770 REM
2780 REM *****************
2790 REM * CAMBIO DE BASE *
2800 REM *****************
2810 REM
2820 PRINT AT 5,18;"1. BASE 2."
2830 PRINT AT 7,18;"2. BASE 10."
2840 PRINT AT 9,18;"3. BASE 16."
2850 LET K$=INKEY$
2860 IF K$="",THEN GOTO 2850
2870 IF K$<>"1" AND K$<>"2" AND K$<>"3" THEN GOTO 2850
2880 LET BAS<(2 AND K$="1")+(10 AND K$="2")+(16 AND K$="3")
2890 LET MOD=VAL(K$)
2900 LET K$=""
2910 RETURN
2920 REM
2930 REM ***************
2940 REM * MUEVE BYTES *
2950 REM ***************
2960 REM
2970 INPUT "DESDE? ";P
2980 IF P<0 OR P>65535  THEN GOTO 2970
2990 INPUT "HASTA? ";D
3000 IF D<0 OR D>65535  THEN GOTO 2990
3010 INPUT "LONGITUD? ";L
3020 IF L<0 OR L>65535! THEN GOTO 3010
3030 IF D<P+L AND D>P THEN POKE 29562,184:RANDOMIZE P+L:POKE 29900,PEEK(23670):
OKE 29901,PEEK(23671):RANDOMIZE D+L:POKE 29902,PEEK(23670):POKE 29903,PEEK(2367
)
3040 IF D<P+L AND D>P THEN RANDOMIZE L:POKE 29904,PEEK(*23670):POKE 29905,PEEK(
3671):RANDOMIZE USR 29550:POKE 29562,176:RETURN
3050 RANDOMIZE P:POKE 29900,PEEK(23670):POKE 29901,PEEK(23671)
3060 RANDOMIZE D:POKE 29902,PEEK(23670):POKE 29903,PEEK(23671)
3070 RANDOMIZE L:POKE 29904,PEEK(23670):POKE 29905,PEEK(23671)
3080 RANDOMIZE USR 29550
3090 RETURN
3100 REM
3110 REM *****************************
3120 REM * PASO DE DECIMAL A BINARIO *
3130 REM *****************************
3140 REM
3150 POKE 29910,VAL(P$)
3160 LET P$=""
3170 FOR D=1 TO 8
3180    LET P$=P$+STR$(USR 29600)
3190 NEXT D
3200 RETURN
3210 REM
3220 REM ******************
3230 REM * CODIGO MAQUINA *
3240 REM ******************
3250 REM
3260 RESTORE 3470
3270 FOR A=0 TO 34
3280    READ B
3290    POKE 29500+A,B
3300 NEXT A
3310 RESTORE 3500
3320 FOR A=0 TO 13
3330    READ B
3340    POKE 29550+A,B
3350 NEXT A
3360 RESTORE 3510
3370 FOR A=0 TO 14
```

```
3380    READ B
3390    POKE 29600+A,B
3400 NEXT A
3410 RETURN
3420 REM
3430 REM *************************************
3440 REM * LINEAS DATA CON EL CODIGO MAQUINA *
3450 REM *************************************
3460 REM
3470 DATA 237,91,176,92,33,204,116,126,213,71,35,26,19,190,32,10
3480 DATA 16,248,209,237,83,176,92,66,75,201,209,19,237,75,176,92
3490 DATA 122,184,40,2,24,222,123,185,200,24,217
3500 DATA 42,204,116,237,91,206,116,237,75,208,116,237,176,201
3510 DATA 58,214,116,203,39,50,214,116,1,0,0,208,14,1,201
3520 REM
3530 REM *****************
3540 REM * INSTRUCCIONES *
3550 REM *****************
3560 REM
3570 CLS
3580 PRINT INVERSE 1;"** I N S T R U C C I O N E S **"
3590 PRINT
3600 PRINT
3610 PRINT FLASH 1;"B";FLASH 0;"USCA BYTES EN MEMORIA"
3620 PRINT FLASH 1;"H";FLASH 0;"ELP. PONE ESTAS INSTRUCCIONES."
3630 PRINT FLASH 1;"J";FLASH 0;"UMP. EJECUTA CODIGO MAQUINA."
3640 PRINT FLASH 1;"L";FLASH 0;"OAD. CARGA BYTES."
3650 PRINT FLASH 1;"M";FLASH 0;"EMORY. AJUSTA EL PUNTERO."
3660 PRINT FLASH 1;"P";FLASH 0;"RINTER. PASA POR IMPRESORA."
3670 PRINT FLASH 1;"S";FLASH 0;"AVE. GRABA BYTES."
3680 PRINT FLASH 1;"T";FLASH 0;"RASLADA BYTES."
3690 PRINT FLASH 1;"X";FLASH 0;" = CAMBIA DE BASE."
3700 PRINT FLASH 1;"5";FLASH 0;" = AUMENTA EL PUNTERO EN 10."
3710 PRINT FLASH 1;"6";FLASH 0;" = AUMENTA EL PUNTERO EN UNO."
3720 PRINT FLASH 1;"7";FLASH 0;" = DISMINUYE EL PUNTERO EN UNO."
3730 PRINT FLASH 1;"8";FLASH 0;" = DISMINUYE EL PUNTERO EN 10."
3740 PRINT FLASH 1;"0";FLASH 0;" = POKEA EN EL PUNTERO."
3750 PRINT #1;FLASH 1;"PULSA UNA TECLA"
3760 PAUSE 0
3770 CLS
3780 RETURN
```

TECNICAS DE ANALISIS

TERMINALES DE PANTALLA (I)



BAJO el nombre de «terminal de pantalla» o «equipo de pantalla» se suele incluir una variedad grande de equipos caracterizados por estar formados por una pantalla de presentación de resultados y, normalmente, un teclado (con o sin memoria auxiliar de almacenamiento temporal de datos y con capacidad de proceso autónomo o sin ella). En algunos casos específicos la pantalla está preparada para aceptar datos por contacto («touch screen») o mediante un lápiz óptico. En fin, el tamaño de la pantalla y el tamaño y disposición del teclado pueden variar, aunque existen ciertos cánones a los que se suelen adecuar las pantallas más comunes. En los ordenadores personales los elementos más visibles son, precisamente, la pantalla y el teclado e, incluso en ocasiones, la propia CPU y los sistemas de almacenamiento están incluidos en la carcasa donde se aloja la pantalla o en la del teclado.

Esta simplicidad que describimos y el abaratamiento de los costes hacen que, hoy en día, no se conciba una aplicación informática que no interactúe con uno o varios terminales de pantalla (aparte, claro está, de los numerosos casos en que la aplicación está concebida, precisamente, para trabajar directamente sobre un ordenador personal o profesional).

Configuraciones

Surgen, por tanto, tres configuraciones posibles para un equipo de pantalla:

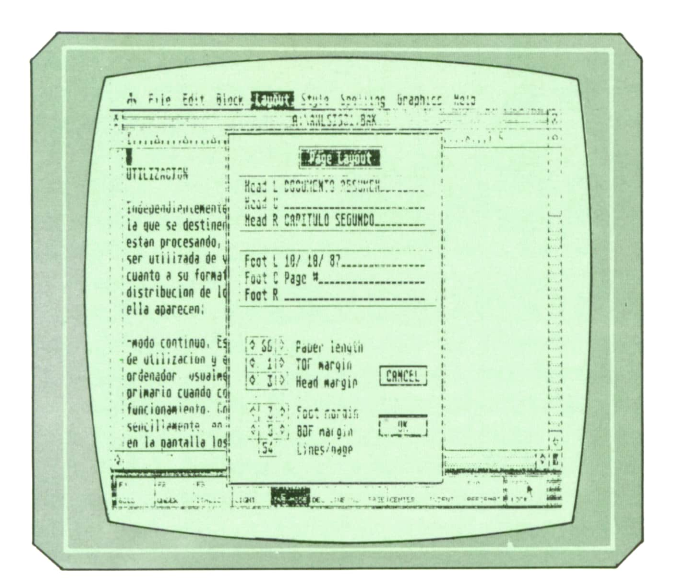
— Terminal de pantalla, propiamente dicho. Se utiliza el vocablo «terminal» cuando el equipo trabaja conectado a un ordenador principal, sin ser el elemento básico de gobierno de dicho ordenador.

— Puesto de trabajo (a veces llamado estación de trabajo, como traducción literal del término inglés «work station»). Se da este nombre a un equipo de pantalla cuando dispone de cierta capacidad de proceso autónomo, aunque, además, esté conectado a un ordenador principal para la realización de ciertas tareas que requieran mayor capacidad de proceso de datos no disponibles en el propio puesto de trabajo, o bien para el almacenamiento posterior de los resultados del proceso.

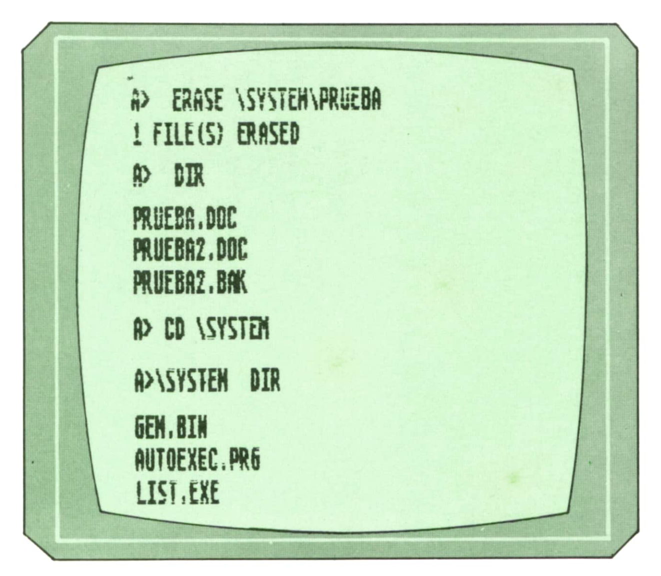
— Ordenador personal. Es una unidad autónoma de trabajo (aunque pueda eventualmente estar conectada a otras, formando una red, pueda utilizarse como estación de trabajo e, incluso, usarse como mero terminal —«inteligente» o no— de un ordenador principal).

Funcionalidad

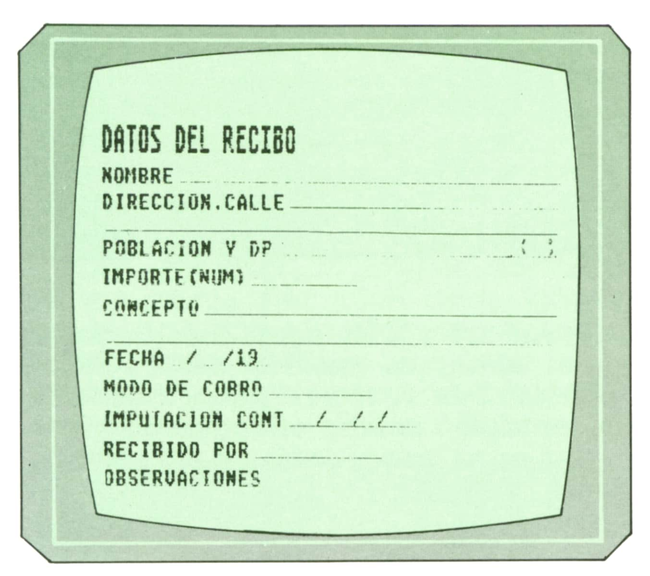
Respecto de la utilidad que se pueda dar a la pantalla del terminal, se pueden considerar varias funcionalidades:




Pantalla de diálogo de un procesador de texto.




Pantalla de gestión de un ordenador personal.




Pantalla de entrada de datos.

— Entrada de datos. En este caso, suelen acumularse numerosos campos en pantalla, tanto porque quien realiza la entrada de los datos está más preparado para operar con una pantalla de estas características, como porque la propia utilidad a obtener así lo exige. Pueden concebirse dos tipos de pantalla de entrada de datos: pasiva (cuando la pantalla meramente recoge los datos y los almacena —a lo sumo con un somero preproceso— para su posterior depuración —en «batch»— o interactiva (si, a medida que se van introduciendo los datos, se van depurando o procesando y, según los resultados del proceso o depuración, la pantalla devuelve información al operador y/o solicita nuevos datos).

— Diálogo. Es una modalidad de presentación interactiva en que priman más los aspectos de tipo conversacional que los de entrada de datos, propiamente dicha. Este es el modo usual de trabajo en los equipos, de diseño, en los sistemas de planificación o toma de decisiones, etc. Las pantallas a utilizar en la modalidad de diálogo suelen tener un aspecto más «descargado» e incluyen, normalmente, información de ayuda a quien está introduciendo datos y contestando las preguntas. Por supuesto, respecto del diálogo en general y de las preguntas en particular hay que tener en cuenta cuanto se comentó en su momento al hablar de los cuestionarios de toma de datos, para hacer la pantalla clara, cómoda de uso y segura (que evite, en la medida de lo posible, la comisión de errores).

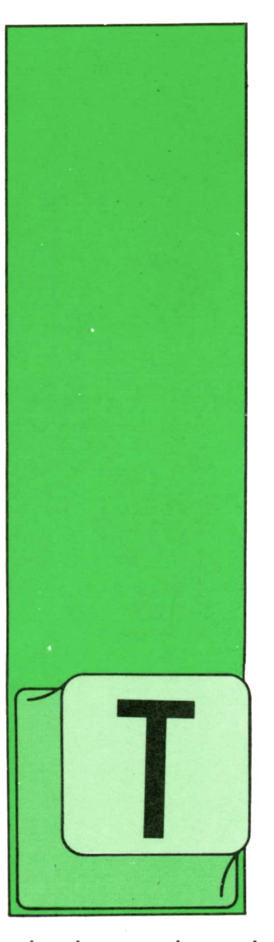
— En otros muchos casos, el terminal de pantalla es el elemento de gobierno del ordenador (ordenadores personales), además de servir como elemento de entrada de datos.

TECNICAS DE PROGRAMACION

PROGRAMACION MODULAR (CONCLUSION)



Funciones recursivas en PASCAL



TAMBIEN el lenguaje PASCAL permite que un módulo sea invocado dentro de su propia definición, es decir, construir módulos recursivos. Veamos, en primer lugar, el mismo ejemplo de la función factorial que dimos en APL:

```
program FAC;
  function factorial (x:integer):real;
  (* Esta función calcula recursivamente
     el factorial de X *)
  begin
    if x<2 then factorial := 1
      else factorial := x*factorial(x-1);
  end;
var numero:integer;
begin
  writeln('Escriba un número');
  readln (numero);
  writeln('Su factorial es = ',factorial
(numero)); end.
```

Su funcionamiento es idéntico al de la función APL descrita al final del capítulo anterior. Veamos algunos ejemplos de su uso:

```
Run
Escriba un número
1
Su factorial es =  1.00000000000000E+000
Run
Escriba un número
5
Su factorial es =  1.20000000000000E+002
Run
Escriba un número
10
Su factorial es =  3.62880000000000E+006
```

Como se verá, hemos tenido que definir el resultado de esta función como REAL, ya que el valor del factorial de un número crece muy de prisa y se sale muy pronto de los límites permitidos para los números enteros. Los valores obtenidos para los factoriales de 1, 5 y 10, dados en el ejemplo, resultan ser 1, 120 y 3628800. Este último ya precisa de la representación interna en forma de número real en la mayor parte de los compiladores de PASCAL.

Esta función sirve únicamente como ejemplo sencillo de recursividad. En la práctica no se usa, pues es mucho más eficiente utilizar un bucle.

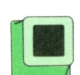
Veamos cómo se ejecuta esta función recursiva, siguiendo con detalle el caso de que queramos calcular el factorial de 3. Al recibir control la función factorial, la variable x vale 3. La única instrucción de la función comprueba primero si x es menor que 2. Como no es este el caso, se ejecutará la parte ELSE, que calcula el resultado de la función multiplicando x (3) por el factorial de x – 1 (2). Es preciso, pues, calcular el factorial de 2. Para ello utilizamos la misma función. Como la variable x es local, el valor antiguo (3) queda fuera del alcance del programa por el momento y la segunda vez que entramos en la función, x valdrá 2. De nuevo ejecutamos la instrucción condicional y

de nuevo tenemos que tomar la alternativa ELSE (pues x no es menor que 2). El resultado del segundo paso por la función factorial es, pues, igual al producto de x (2) por el factorial de x – 1 (1). Hay que calcular, por tanto, el factorial de 1, para lo cual entramos por tercera vez en la función. De nuevo la variable x es local, y ahora vale 1. Esta vez, al ejecutar la instrucción condicional, seguiremos la alternativa THEN, pues x (1) es menor que 2. El resultado de la función es, por tanto, 1.

Vamos ahora a deshacer el camino andado. Ya sabemos que factorial (1) es igual a 1. Salimos, por tanto, de la tercera ejecución de la función y pasamos a la segunda. La variable local x recupera ahora su valor 2. La función se quedó interrumpida cuando iba a multiplicar x por factorial (1). Ahora podemos calcular este producto, que es igual a 2 por 1, es decir, 2. Ya conocemos el valor de factorial (2) y podemos abandonar la segunda ejecución de la función. Pasamos a la primera, donde la variable local x recupera su valor 3. Esta función había quedado interrumpida cuando iba a multiplicar x por factorial (2). Ahora ya conocemos ambos valores y el producto es igual a 3 por 2, es decir, 6. Salimos, por tanto, de la primera ejecución de la función factorial, obteniendo el siguiente resultado final: factorial (3) es igual a 6.

Resumamos lo anterior en una tabla:

PASO FUNCION	VALOR DE X	X<2	RESULTADO
1	3	NO	3*FACTORIAL(2)
2	2	NO	2*FACTORIAL(1)
3	1	SI	1
2	2		2*1=2
1	3		3*2=6



Funciones recursivas en BASIC

Es posible construir funciones recursivas en BASIC, aunque su programación es más molesta y difícil que en APL o PASCAL, donde la existencia de variables locales facilita considerablemente las cosas. Veamos cómo se programaría en BASIC de forma recursiva la funcional FACTORIAL que hemos tomado como ejemplo:

```
10 REM Función recursiva en BASIC
20 DIM P(175)
30 PRINT "Escriba un número"
40 INPUT X
50 I=1
60 GOSUB 1000
70 PRINT "Su factorial es =";F
80 END
1000 REM Cálculo recursivo del factorial
1010 IF X<2 THEN F=1:RETURN
1020 P(I)=X
1030 I=I+1
1040 X=X-1
1050 GOSUB 1000
1060 I=I-1
1070 F=F*P(I)
1080 RETURN
```

Veamos que, efectivamente, funciona:

```
Run
Escriba un número
? 1
Su factorial es =  1
Run
Escriba un número
? 5
Su factorial es =  120
Run
Escriba un número
? 10
Su factorial es =  3628800
```

Puede comprobarse que los resultados son los mismos que en el caso de PASCAL. Pero observemos la función recursiva: es evidente que se llama a sí misma (la instrucción GOSUB 1000 puede ejecutarse después de llamar al módulo de etiqueta 1000 y antes de devolver el control a quien lo llamó). En el momento de recibir control el módulo, la variable X contiene el valor cuyo factorial deseamos calcular. El resultado se devuelve en la variable F. Sin embargo, como no existen variables locales, tenemos que guardar en algún sitio los valores que va tomando la variable X en cada llamada al módulo. Para ello utilizaremos la variable vectorial P, que podrá tener hasta 175 elementos (en la mayor parte de los ordenadores personales es imposible calcular el factorial de un número mayor de 175, pues el resultado sería demasiado grande). Tendremos que definir también un contador para señalar hasta dónde hemos llenado el vector P. Este contador será la variable I, que inicialmente vale

1, porque la primera posición no utilizada de P es la primera.

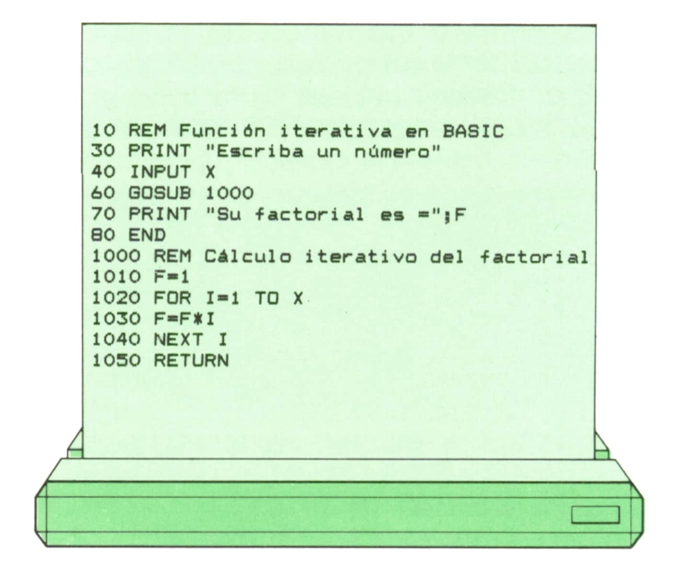
Cada vez que reciba control el módulo de etiqueta 1000, empezará por comprobar si el valor de X es menor que 2. En caso afirmativo, hará F (el resultado) igual a 1 y devolverá control. En caso contrario, guardará el valor de X en la primera posición libre de la variable vectorial P, aumentará en una unidad el valor de I, puesto que ahora la primera posición libre es la siguiente, reducirá el valor de X en una unidad, pues va a calcular el factorial de X – 1, y llamará al módulo de etiqueta 1000. Después de obtener el factorial de X – 1, este valor debe multiplicarse por el valor antiguo de X al entrar en el módulo. Para obtenerlo bastará con disminuir en una unidad el valor de I y sacarlo de la variable vectorial P (I). Este número se multiplica por F y se coloca en F, pues es el resultado de esta pasada de cálculo de la función factorial. Ahora ya podemos devolver el control al último que llamó al módulo de etiqueta 1000, que pudo ser el mismo desde la instrucción de etiqueta 1050.

La tabla siguiente detalla cómo va ejecutándose el programa:

INSTRUCCION	VALOR DE X	I	F	P
40 LEE X	3			
50	3	1		
60	3	1		
1000-1010	3	1		
1020	3	1		3
1030	3	2		3
1040	2	2		3
1050	2	2		3
1000-1010	2	2		3
1020	2	2		3 2
1030	2	3		3 2
1040	1	3		3 2
1050	1	3		3 2
1000	1	3		3 2
1010 RETURN	1	3	1	3 2
1060	1	2	1	3 2
1070	1	2	2	3 2
1080 RETURN	1	2	2	3 2
1060	1	1	2	3 2
1070	1	1	6	3 2
1080 RETURN	1	1	6	3 2
70 IMPRIME F			6	

Se observan claramente en la tabla las tres pasadas del módulo que calcula la función factorial y los valores que van tomando las variables a lo largo de la ejecución del programa. Como se observará, el proceso es muy semejante al explicado para las versiones APL y PASCAL de este programa, pero en BASIC tenemos el problema adicional de tener que guardar los valores sucesivos de las variables que han de funcionar como locales. En este caso concreto sólo había una, pero es muy posible que se desee construir funciones recursivas con muchas variables locales, y en tal caso el lenguaje BASIC es totalmente desaconsejable.

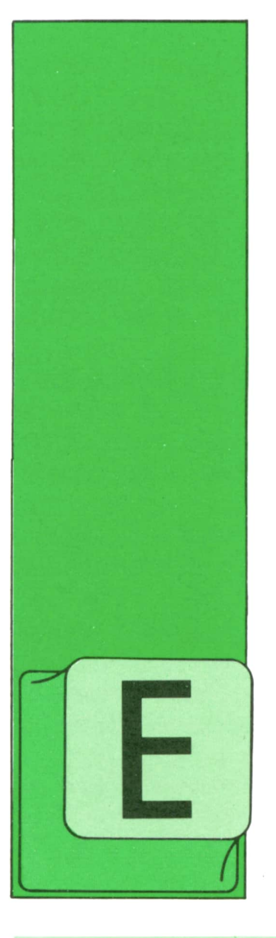
El ejemplo que hemos seguido, la función factorial, es un caso muy sencillo y ha sido útil para aclarar el funcionamiento de la programación modular recursiva. Pero no es conveniente utilizarlo en la práctica, por razones de eficiencia. Ya hemos visto que esto es absolutamente innecesario en APL, pues en este lenguaje existe un símbolo que calcula directamente esta operación matemática. En PASCAL y BASIC sería más conveniente utilizar un procedimiento iterativo (con bucles). Veamos el caso de BASIC:



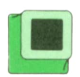
```
10 REM Función iterativa en BASIC
30 PRINT "Escriba un número"
40 INPUT X
60 GOSUB 1000
70 PRINT "Su factorial es =";F
80 END
1000 REM Cálculo iterativo del factorial
1010 F=1
1020 FOR I=1 TO X
1030 F=F*I
1040 NEXT I
1050 RETURN
```

APLICACIONES

DBASE III (y 2)



EN esta segunda parte dedicada el dBASE III estudiaremos los comandos más usuales del dBASE y realizaremos una aplicación práctica con ellos.



Comando de manejo de ficheros

— CREATE: Permite crear un fichero de base de datos.

— USE: Selecciona un fichero para su utilización.

```
                                                          MANDATOS INICIALES
 - AYUDA MAXIMA -

                Mandatos dBASE III --- Conjunto Inicial
                ========================================

 1 - ?                 11 - DELETE FILE        21 - LIST           31 - SEEK
 2 - APPEND            12 - DIR                22 - LOCATE         32 - SET
 3 - AVERAGE           13 - DISPLAY            23 - MODIFY         33 - SKIP
 4 - BROWSE            14 - DO                 24 - PACK           34 - SORT
 5 - CHANGE            15 - EDIT               25 - QUIT           35 - STORE
 6 - CLEAR             16 - ERASE              26 - RECALL         36 - SUM
 7 - COPY              17 - FIND               27 - RELEASE        37 - TOTAL
 8 - COUNT             18 - GO/GOTO            28 - RENAME         38 - TYPE
 9 - CREATE            19 - INDEX              29 - REPLACE        39 - USE
10 - DELETE            20 - LABEL              30 - REPORT

 PgUp=pantalla previa, Esc=salir de AYUDA,  ^Home=menú previo o teclear mandato

                       Teclee >
```



Fig. 1. Mandatos iniciales de DBASE.

Los comandos pueden agruparse por sus funciones en:

— Comandos de manejo del fichero.

— Comandos de localización de datos.

— Comandos de entorno.

— Comandos de usos generales.

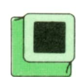
— APPEND: Añade registros al final de un fichero de datos.

— INSERT: Inserta registros en alguna posición específica.

— DELETE: Borra registros.

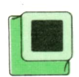
— PACK: Elimina físicamente los registros borrados.

— RECALL: Recupera los registros borrados.
— ERASE: Borra ficheros.
— INDEX: Crea un índice de un fichero, permitiendo reducir el tiempo de búsqueda de una forma considerable.
— SORT: Ordena alfabéticamente o numéricamente los ficheros.



Comandos de localización de datos

— LOCATE: Busca datos de un fichero según una expresión lógica.
— FIND: Busca datos en un fichero indexado.



Comandos de entorno

Todos estos comandos tienen dos posibilidades: activados y desactivados (on/off), mostraremos solamente los más representativos entre los muchos que tiene el programa:

SET CONSOLE: Permite o no que las salidas de datos se produzcan por la pantalla.

SET DEBUG: Cuando está activado entra en el modo de depuración de programas, de forma que cuando ejecuta una instrucción se realiza una parada.

SET EXACT: Obliga, si está activado, que en las búsquedas la cadena buscada coincida exactamente con la encontrada en el registro leído.

SET MENUS: Muestra los menús de ayuda o no.

SET UNIQUE: Si está activado (on), en los ficheros indexados no se permiten claves duplicadas.

SET PRINT: Cuando se activa todo lo enviado a la pantalla se envía también a la impresora.

Realizaremos ahora un ejemplo práctico con el dBASE III:

Supongamos que queremos llenar una agenda telefónica para poder consultar datos.

El primer paso será crear un fichero de datos con el comando CREATE.

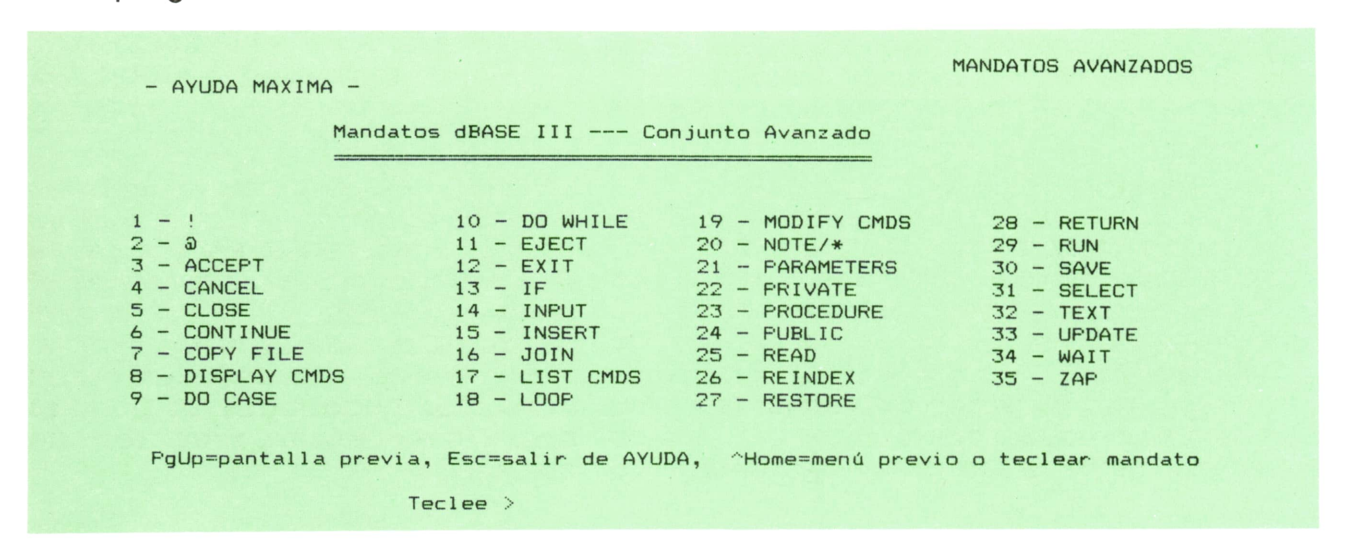




Fig. 2. Mandatos avanzados de DBASE.

El comando completo es: CREATE AGENDA; donde agenda es el nombre que tendrá el fichero.

Pasamos después a introducir su estructura. Una vez creado el fichero el siguiente paso es introducir los datos en él, para lo cual ponemos:

USE AGENDA
APPEND

Para terminar de introducir datos pulsemos el RETURN al principio del primer campo.

Para localizar los datos en este fichero podremos utilizar el comando LOCATE, primeramente USE AGENDA, si es que no lo hemos hecho antes.

El formato de la instrucción LOCATE es:

LOCATE FOR descripción

Por ejemplo, si queremos encontrar los datos de JOSE PEREZ pondremos:

LOCATE FOR NOMBRE = " JOSE PEREZ "

Esta instrucción en el programa se habrá situado en el registro que cumple esa expresión, pero sin mostrar su contenido; para ello bastará con poner DISP, apareciendo los datos del registro buscado.

Esta búsqueda la podemos realizar con cualquier campo del fichero, en nuestro ejemplo con la dirección o el teléfono.

Para realizar el listado del fichero utilizaremos la opción REPORT. Como paso previo tendremos que crear el formato del listado con el comando: CREATE REPORT. Con el formato ya creado utilizaremos REPORT nombre, y si queremos que el listado salga por impresora, pondremos entonces: REPORT nombre TOPRINT.

```
                                                          SET ON/OFF
    - AYUDA MAXIMA -

                          Mandatos SET ON/OFF
                          ===================

  1 - SET ALTERNATE        8 - SET DELIMITER      15 - SET INTENSITY
  2 - SET BELL             9 - SET ECHO           16 - SET MENUS
  3 - SET CARRY           10 - SET ESCAPE         17 - SET PRINT
  4 - SET CONFIRM         11 - SET EXACT          18 - SET SAFETY
  5 - SET CONSOLE         12 - SET FIXED          19 - SET STEP
  6 - SET DEBUG           13 - SET HEADING        20 - SET TALK
  7 - SET DELETED         14 - SET HELP           21 - SET UNIQUE

        El valor por omisión (ON u OFF) se muestra en mayúsculas.

  PgUp=pantalla previa, Esc=salir de AYUDA,  ^Home=menú previo o teclear mandato

                      Teclee >
```


Fig. 3. Mandatos set ON/OFF. Teclee >

Comando ASSIST

Por medio de este comando el programa dBASE proporciona una potente ayuda para todos aquellos usuarios que no estén familiarizados con el programa.

Al teclear ASSIST el programa entra en un primer menú, en el que explica las teclas de función que existen.

Posteriormente el programa muestra las primeras opciones que se activan simplemente poniendo el cursor encima de ellas y pulsando el RETURN. Una vez se ha seleccionado una de las opciones, el programa va guiando paulatinamente al usuario, de tal forma que no es necesario recordar ningún comando, ya que el mismo va mostrando todas las posibles opciones y la secuencia correcta en que hay que ir introduciéndolas.

```
    - AYUDA MAXIMA -                                            FUNCIONES
                          Funciones de dBASE III
                          ======================

  1 - ASC()        10 - DAY()          19 - LOG()          28 - SPACE()
  2 - AT()         11 - DELETED()      20 - LOWER()        29 - SQRT()
  3 - BOF()        12 - DOW()          21 - MACRO/&        30 - STR()
  4 - CDOW()       13 - DTOC()         22 - MONTH()        31 - SUBSTR()
  5 - CHR()        14 - EOF()          23 - PCOL()         32 - TIME()
  6 - CMONTH()     15 - EXP()          24 - PROW()         33 - TRIM()
  7 - COL()        16 - FILE()         25 - RECNO()        34 - TYPE()
  8 - CTOD()       17 - INT()          26 - ROUND()        35 - UPPER()
  9 - DATE()       18 - LEN()          27 - ROW()          36 - VAL()
                                                           37 - YEAR()

 PgUp=pantalla previa, Esc=salir de AYUDA,  ^Home=menú previo o teclear mandato Teclee >
```


Fig. 4. Funciones de DBASE. Teclee >

SET TO

- AYUDA MAXIMA -

Mandatos SET TO

1 - SET ALTERNATE	8 - SET FUNCTION
2 - SET COLOR	9 - SET FORMAT
3 - SET DECIMALS	10 - SET INDEX
4 - SET DEFAULT	11 - SET MARGIN
5 - SET DELIMITER	12 - SET PATH
6 - SET DEVICE	13 - SET PROCEDURE
7 - SET FILTER	14 - SET RELATION

PgUp=pantalla previa, Esc=salir de AYUDA, ^Home=menú previo o teclear mandato

Teclee >



Fig. 5. Mandatos set de DBASE.

De esta forma tan sencilla el programa permite que el usuario utilice las funciones más avanzadas del programa, guiándolo a través de todas las opciones posibles.

HELP

Con este comando el programa muestra información sobre cualquier aspecto del programa, bien sean comandos, funciones u otras opciones.

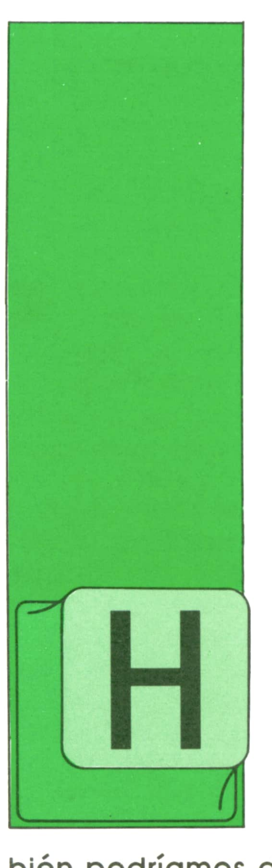
El comando HELP se activa automáticamente cuando el usuario introduce un comando incorrecto, preguntando en este caso si desea ayuda. Otra forma de activar el comando es tecleando HELP, con la que el programa entrará en el modo de ayuda.

Mandatos dBASE III - Conjunto inicial

1 — ?	11 — DELETE FILE	21 — LIST	31 — SEEK
2 — APPEND	12 — DIR	22 — LOCATE	32 — SET
3 — AVERAGE	13 — DISPLAY	23 — MODIFY	33 — SKIP
4 — BROWSE	14 — DO	24 — PACK	34 — SORT
5 — CHANGE	15 — EDIT	25 — QUIT	35 — STORE
6 — CLEAR	16 — ERASE	26 — RECALL	36 — SUM
7 — COPY	17 — FIND	27 — RELEASE	37 — TOTAL
8 — COUNT	18 — GO/GOTO	28 — RENAME	38 — TYPE
9 — CREATE	19 — INDEX	29 — REPLACE	39 — USE
10 — DELETE	20 — LABEL	30 — REPORT	

PgUp = pantalla previa, Esc = salir de AYUDA, ^Home = menú previo o teclear mandato, Teclee >

PASCAL





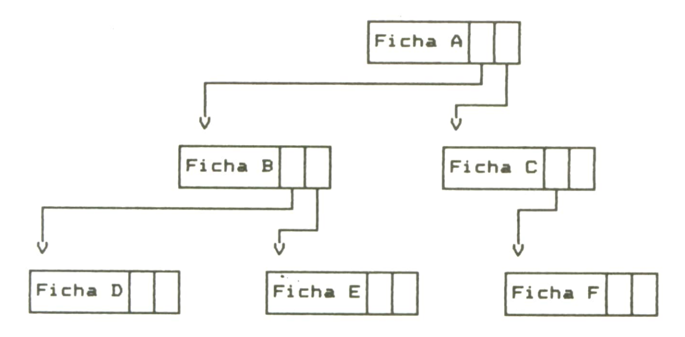
Estructuras árbol

HEMOS visto que una lista encadenada es una secuencia de elementos en que cada uno contiene la referencia necesaria para acceder al elemento siguiente. También podríamos definirla recursivamente de la siguiente manera:

Una lista de tipo base T es:

1. Una lista vacía.
2. O bien el encadenamiento de un elemento de tipo T con una lista de tipo base T (y vuelta a empezar).

Supongamos ahora que los elementos con los que, en principio, íbamos a formar una lista lineal contengan más de un campo donde guardar punteros a elementos similares; podríamos tener una estructura como ésta:



A estructuras semejantes se les denomina «estructuras árbol»; su definición recursiva es la siguiente:

Una estructura árbol de tipo base T es:

1. Una estructura vacía.
2. O bien un elemento de tipo T del que «cuelgan» un cierto número de estructuras árbol de tipo base T, sin elementos comunes entre sí; a estas estructuras se les denomina «subárboles».

El árbol de la figura está formado, por tanto, por el elemento A seguido del subárbol encabezado por el elemento B y del subárbol encabezado por C. Análogamente, podemos decir que el primero de éstos está formado por el elemento B seguido de los subárboles encabezados por D y E, y que el segundo está formado por el elemento C seguido del subárbol encabezado por el elemento F y de otro subárbol vacío.

Comparando las definiciones de árbol y lista, resulta evidente que esta última es un caso particular de árbol en que cada elemento tiene a lo sumo un subárbol.

Veamos a continuación una serie de términos habituales al trabajar con árboles.

Se denomina «nodo» a cada elemento del árbol. Cuando un nodo Y depende directamente de un nodo X se dice que Y es «descendente» de X e, inversamente, que X es el «antecesor» de Y (recordemos los árboles genealógicos); en el árbol del ejemplo, A sería el antecesor de B y C, y B tendría a D y E como descendentes.

El nodo superior, del que dependen todos los demás, se denomina «raíz» (A en el ejemplo), mientras que aquéllos que no tienen descendentes se denominan nodos terminales u «hojas» (de estos últimos se podría decir, siguiendo la definición recursiva, que son árboles que constan de un elemento del que cuelgan dos subárboles vacíos).

Por definición, se dice que la raíz de un árbol está en el «nivel» 1, sus descendentes

en el 2, los de éstos en el 3, etc.; el máximo nivel encontrable en un árbol se dice que es su profundidad o altura (la altura del árbol del ejemplo sería 3).

El número de descendentes directos de un nodo es su «grado»; el máximo de los grados de todos los nodos de un árbol es el grado del árbol (el árbol del ejemplo sería, por tanto, de grado 2).

Un «árbol ordenado» es aquél en que los descendentes de cada nodo están ordenados según un criterio dado; por ejemplo, los árboles de la figura:



están ordenados según criterios distintos y son desde ese punto de vista, por tanto, distintos. En otras palabras, un árbol está ordenado cuando las diferentes «ramas» que salen de cada nodo tienen nombre propio y no pueden intercambiarse entre sí por las buenas.

Los árboles ordenados de grado 2 se denominan árboles «binarios» y tienen una gran importancia en proceso de datos.

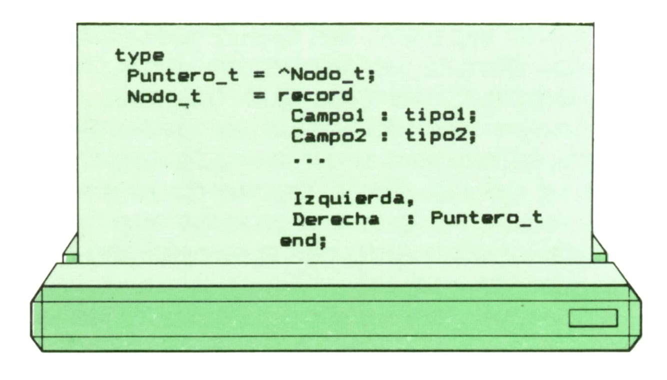


Arboles binarios

Se puede definir un árbol binario como:

1. Un conjunto vacío de elementos.
2. O bien un conjunto formado por un nodo con dos árboles binarios disjuntos, llamados subárbol «izquierdo» y «derecho» del nodo en cuestión.

El árbol del ejemplo, si lo consideramos ordenado, es un árbol binario. La definición de cada nodo podría ser:



```
type
 Puntero_t = ^Nodo_t;
 Nodo_t    = record
              Campo1 : tipo1;
              Campo2 : tipo2;
              ...

              Izquierda,
              Derecha   : Puntero_t
             end;
```

Los campos Izquierda y Derecha contendrían punteros para acceder a los dos posibles descendentes de cada nodo; la ausencia de éstos se indicaría con el valor predefinido NIL. Se necesitaría, además, una variable de tipo Puntero -t, que podríamos llamar Raiz, para acceder a la raíz del árbol.

Si necesitásemos recorrer todos los nodos de un árbol para hacer algo con ellos (por ejemplo, mostrar su contenido), podríamos hacer lo siguiente: visitar primero el nodo raíz, luego su descendente izquierdo y entonces... ¿con cuál seguir?

Si seguimos con los descendentes del izquierdo, ¿cuándo procesaremos el derecho y sus descendentes?; habrá que anotar en algún sitio que están pendientes de visitar; el problema se volverá a presentar con los dos descendentes del izquierdo, etc. Por otra parte, si tras visitar el izquierdo pasamos al derecho, habrá que anotar que los descendentes del izquierdo están pendientes de ser visitados... Además, cuando queramos pasar a visitar los nodos pendientes, habrá que rehacer todo el camino desde la raíz para llegar a ellos.

Afortunadamente, los árboles, y no sólo los binarios, son un caso típico de estructuras de datos que, teniendo una definición recursiva natural, se pueden manejar con procedimientos recursivos de una manera muy sencilla. Por ejemplo, para recorrer un árbol el procedimiento podría ser:

«Recorrer el árbol encabezado por el nodo apuntado por tal puntero:»

1. Hacer lo necesario con el nodo en cuestión (mostrarlo...).

2. Si tiene descendente izquierdo, entonces «recorrer el árbol encabezado por el nodo apuntado por Izquierdo».

3. Si tiene descendente derecho, entonces «recorrer el árbol encabezado por el nodo apuntado por Derecho».

Para recorrer el árbol completo tendríamos que «recorrer el árbol encabezado por el nodo apuntado por Raiz». Se podría programar así:

```
procedure Recorrer (P: Puntero_t);
 begin
  with P^ do
    begin
     Procesar (P^);
     if Izquierdo <> nil then Recorrer
     (Izquierdo);
     if Derecho   <> nil then Recorrer
     (Derecho)
    end
 end;
```

Procesar sería el procedimiento que se desease aplicar a los diferentes nodos. También podríamos escribir el procedimiento de recorrido así:

```
procedure Recorrer (P: Puntero_t);
 begin
  if P <> nil then with P^ do
    begin
     Procesar (P^);
     Recorrer (Izquierdo);
     Recorrer (Derecho)
    end
 end;
```

es decir, llamar al procedimiento cualquiera que sea el valor de Izquierda y Derecha, que ya se encargará él de comprobarlo. De esta manera, además, podremos ejecutar Recorrer (Raiz) sin comprobar antes el valor de Raiz.

Con estos procedimientos recorreríamos cada nodo antes que sus dos subárboles; se le denomina por ello recorrido en «pre-orden».

Aplicado al árbol del ejemplo, la secuencia de recorrido sería A, B, D, E, C y F.

Si el procedimiento fuese:

```
procedure Recorrer (P: Puntero_t);
 begin
  if P <> nil then with P^ do
    begin
     Recorrer (Izquierdo);
     Procesar (P^);
     Recorrer (Derecho)
    end
 end;
```

antes de cada nodo recorreríamos primero uno de los subárboles (en este caso concreto el izquierdo, pero podría ser el otro), luego el nodo en cuestión y, por último, el otro subárbol. Sería un recorrido en «orden central»; aplicado al ejemplo, la secuencia sería D, B, E, A, F y C.

Por último, con el procedimiento:

```
procedure Recorrer (P: Puntero_t);
 begin
  if P <> nil then with P^ do
    begin
     Recorrer (Izquierdo);
     Recorrer (Derecho);
     Procesar (P^)
    end
 end;
```

tendríamos un recorrido en «post-orden» que, aplicado al ejemplo, daría la secuencia D, E, B, F, C y A.

Si representásemos el árbol del ejemplo al estilo de un diagrama sináptico, podríamos tener tres formas básicas:

Observando de arriba hacia abajo en qué orden quedan los diferentes nodos, salta a la vista que cada una de las figuras corresponde a uno de los tres órdenes posibles de recorrido.

Un programa para representar de alguna de estas formas el contenido de un árbol en la pantalla del ordenador o en una impresora, debería, al recorrer los nodos, calcular en todo momento el nivel de cada uno para así saber cuánto avanzar hacia la derecha antes de escribir.

Un procedimiento de recorrido en orden central teniendo disponible en todo momento el nivel de cada nodo podría ser:

```
procedure Recorrer (P: Puntero_t;
N: integer);
(*---------------------------*)
(*  N contiene el nivel del  *)
(*  nodo apuntado por P.     *)
(*---------------------------*)
 begin
  if P <> nil then with P^ do
    begin
     Recorrer (Izquierdo, N + 1);
     Procesar (P^);
     Recorrer (Derecho, N + 1)
    end
 end;
```

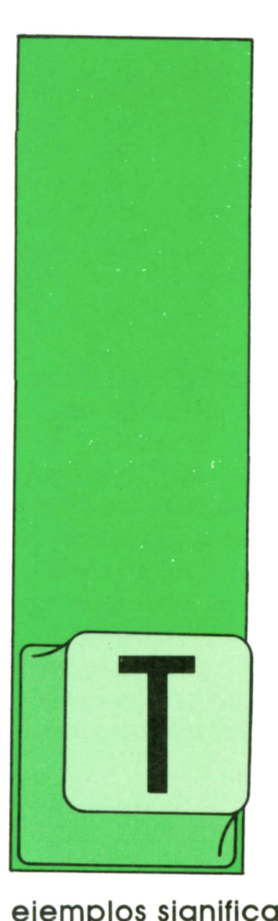
Si el nivel de un nodo dado es N, el nivel de cualquiera de sus descendentes es N + 1. Para empezar a recorrer el árbol ejecutaríamos:.

Recorre (Raiz, 1);

En caso de que el procedimiento Procesar lo necesitase, podría modificarse para transferirle en la lista de parámetros el valor del nivel del nodo a procesar.

OTROS LENGUAJES

ADA-4





Sintaxis del lenguaje ADA (continuación)

TRATAREMOS en este último capítulo de explicar brevemente las diferencias restantes con respecto a otros lenguajes más conocidos. Daremos también algunos ejemplos significativos.

Una de las características del lenguaje ADA que es muy poco usual en los lenguajes existentes actualmente es la posibilidad de la ejecución de tareas en paralelo. Estas tareas pueden ser implementadas en ordenadores con un solo procesador (suele ser lo más habitual), o en sistemas con múltiples procesadores. Esta última posibilidad, la cual abre una nueva dimensión en la programación, nunca utilizada hasta ahora, tiene una serie de dificultades, esenciales a este tipo de procesamiento, que los lenguajes como ADA, y en menor medida MODULA-2, tratan de solventar.

Para ello, el ADA trata las tareas como entidades independientes, pero además dispone de los mecanismos adecuados para que estas tareas puedan ser coordinadas adecuadamente y sincronizadas eficientemente.

La forma de definir una tarea (task) es prácticamente la misma que la que se utiliza en MODULA-2 para definir un procedimiento dentro de un módulo, aunque las palabras clave sean diferentes.

Veamos un ejemplo:

```
task PROTECTED_ARRAY is
   -- INDEX and ELEM are global types
   entry READ (N : in INDEX; V : out ELEM);
   entry WRITE(N : in INDEX; E : in  ELEM);
end;

task body PROTECTED_ARRAY is
   TABLE : array(INDEX) of ELEM := (INDEX => 0);
begin
   loop
      select
         accept READ (N : in INDEX; V : out ELEM) do
            V := TABLE(N);
         end READ;
      or
         accept WRITE(N : in INDEX; E : in  ELEM) do
            TABLE(N) := E;
         end WRITE;
      end select;
   end loop;
end PROTECTED_ARRAY;
```

Con el propósito de sincronizar las tareas entre sí, el ADA dispone de sentencias de retardo (delay) y de tipos de variables temporales (time) basados en el tipo básico «duration».

El ADA también dispone de sentencias de espera selectiva, con las cuales se pueden hacer varias combinaciones, muy útiles para la sincronización.

Las llamadas condicionales a sentencias son también posibles si éstas están disponibles inmediatamente.

Cada tarea (task) tiene en el ADA un nivel de prioridad que se define explícitamente en el especificador de la misma. Este nivel permite la jerarquización de tareas. Esto es muy difícil, ya que el ADA ofrece numerosas utilidades que podríamos denominar de bajo nivel, como es la activación de una tarea mediante una interrupción. Al estar jerarquizadas las tareas, habrá interrupciones que provoquen la ejecución de sus tareas y otras que no, ya que al ser de nivel con menor prioridad, no podrán interrumpir a las que están en ejecución.

En ADA existen también sentencias dedicadas al tratamiento de las situaciones catastróficas, como situaciones de «abort», que permiten el tratamiento software de errores de todo tipo, incluso hardware.

También existen las variables solapadas (shared), que son accesibles desde dos o más tareas diferentes pero no desde otras. Esto permite el paso de parámetros entre tareas de una forma dinámica.

La compilación separada está también dotada de numerosas utilidades que permiten hacer diferentes tratamientos, con cada unidad compilable, como las cláusulas WITH, en compilación.

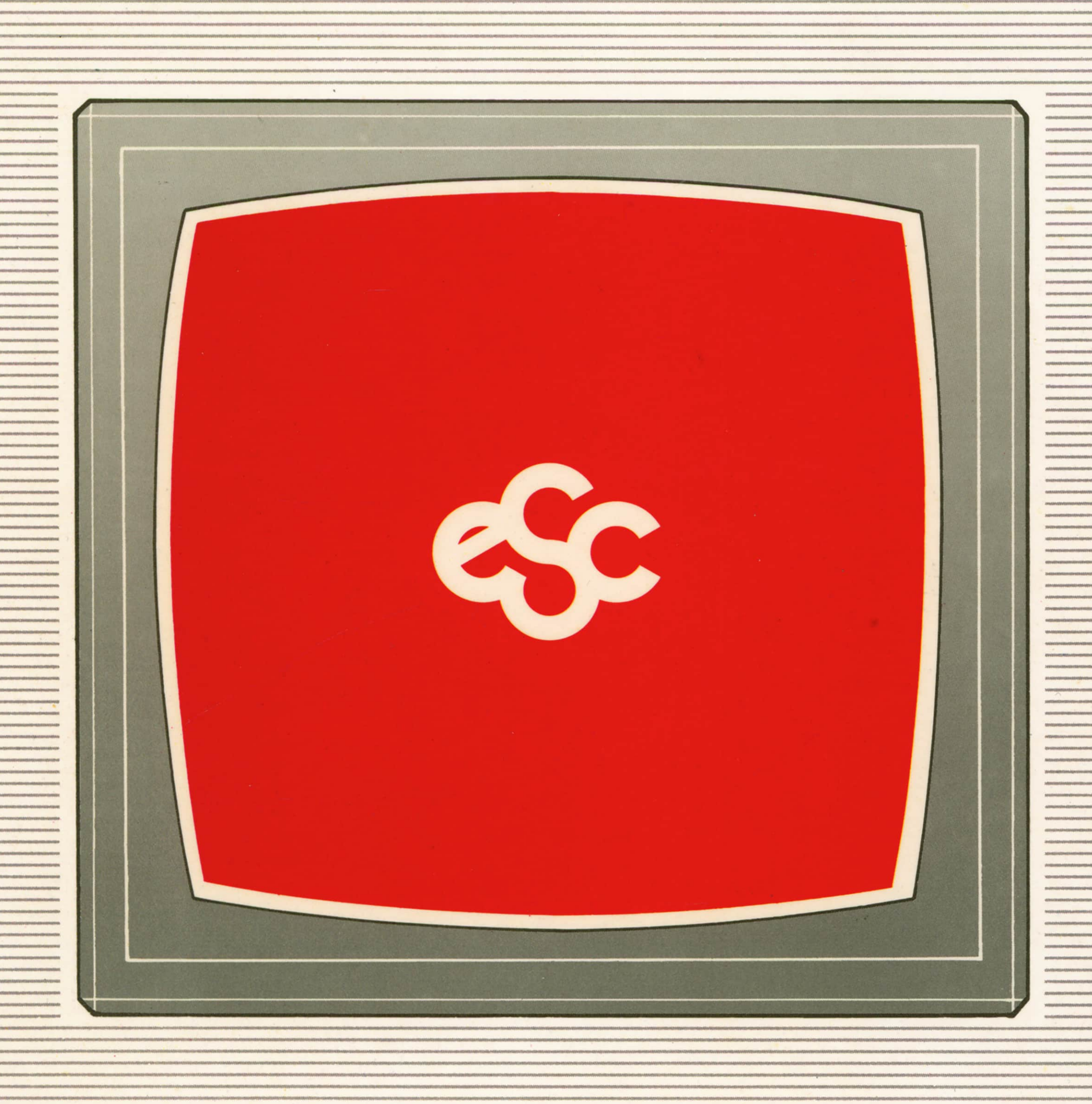
Dentro de las unidades de compilación es posible tambien la jerarquización mediante la utilización de subunidades, otro conjunto de instrucciones disponible en ADA.

La compilación mediante un orden preestablecido (a elegir) y la selección detallada de las librerías a utilizar, así como de otras utilidades más sofisticadas, permiten que el código objeto resultado de una compilación en ADA pueda resultar muy optimizado.

Por último, diremos que el ADA está especialmente diseñado para la manipulación de todo tipo de excepciones, por lo que puede utilizarse como un lenguaje para el diseño de sistemas informáticos, no sólo de aplicaciones.

Como resumen, diremos que el lenguaje ADA es un lenguaje de los más potentes, pero tiene el gran defecto de un número muy grande de instrucciones diferentes, lo que hace que para algunos tenga una estructura demasiado complicada.

Este lenguaje será pronto operativo en sistemas de microordenador, debido sobre todo al rápido incremento en la potencia de cálculo y en la capacidad de memoria que están teniendo últimamente los ordenadores personales.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼